EFFICIENT MOBILE MULTIMEDIA STREAMING

by

Cheng-Hsin Hsu

M.Eng., University of Maryland, College Park, Maryland, USA, 2003

M.Sc., National Chung-Cheng University, Chia-Yi, Taiwan, 2000

B.Sc., National Chung-Cheng University, Chia-Yi, Taiwan, 1996

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Fall 2009

APPROVAL

Name: Cheng-Hsin Hsu

Degree: Doctor of Philosophy

Title of Thesis: Efficient Mobile Multimedia Streaming

Examining Committee: Dr. Joseph Peters
Chair

Dr. Mohamed Hefeeda, Senior Supervisor

Dr. Ramesh Krishnamurti, Supervisor

Dr. Funda Ergun, SFU Examiner

Dr. Charles Krasic, External Examiner,
Assistant Professor of Computer Science,
University of British Columbia

Date Approved: November 24, 2009

Declaration of
Partial Copyright Licence

Abstract

Modern mobile devices have evolved into small computers that can render multimedia streaming content anywhere and anytime. These devices can extend the viewing time of users and provide more business opportunities for service providers. Mobile devices, however, make a challenging platform for providing high-quality multimedia services. The goal of this thesis is to identify these challenges from various aspects, and propose efficient and systematic solutions to solve them. In particular, we study mobile video broadcast networks in which a base station concurrently transmits multiple video streams over a shared air medium to many mobile devices. We propose algorithms to optimize various quality-of-service metrics, including streaming quality, bandwidth efficiency, energy saving, and channel switching delay. We analytically analyze the proposed algorithms, and we evaluate them using numerical methods and simulations. In addition, we implement the algorithms in a real testbed to show their practicality and efficiency. Our analytical, simulation, and experimental results indicate that the proposed algorithms can: (i) maximize energy saving of mobile devices, (ii) maximize bandwidth efficiency of the wireless network, (iii) minimize channel switching delays on mobile devices, and (iv) efficiently support heterogeneous mobile devices. Last, we give network operators guidelines on choosing solutions suitable for their mobile broadcast networks, which allow them to provide millions of mobile users much better viewing experiences, attract more subscribers, and thus increase the revenues.

Keywords: video streaming; mobile video; broadcast networks; transmission scheduling; scalable video coding

Acknowledgments

I am heartily grateful to my senior supervisor, Dr. Mohamed Hefeeda, whose encouragement, guidance and support from the initial to the final steps enabled me to develop research skills and an understanding of the exciting fields of multimedia networking and distributed systems. He always made his time available for me and provided valuable advice during my graduate career. This thesis would not have been possible without him.

I owe my deepest gratitude to Dr. Ramesh Krishnamurti, my supervisor, for too many insightful discussions during the past few years. His experience in solving optimization problems inspired me and enabled me to think and address the research problems in multimedia systems from different angles. I would like to show my gratitude to Dr. Funda Ergun, my thesis examiner, and Dr. Charles Krasic, my external thesis examiner, for being on my committee and reviewing this thesis. I would also like to thank Dr. Joseph Peters for taking the time to chair my thesis defense.

I would like to thank all my colleagues at Network Systems Lab for their support and help. It was my honor to work with these talented people. I am especially grateful to Osama Saleh, Behrooz Noorizadeh, Kianoosh Mokhtarian, Ahmed Hamza, Cong Ly, Yi Liu, and Yuanbin Shen.

I am indebted to my family for their love, encouragement, and endless support. I owe my loving thanks to my wife Huei-Ying, my sons Ryan and Charlie. They have lost a lot during my graduate career at SFU. Without their encouragement and understanding it would be impossible for me to finish this thesis. This thesis is dedicated to them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we introduce mobile video broadcast networks and highlight important performance metrics in them. We describe the challenges of providing high-quality streaming services in these networks, and we present several research problems to overcome these challenges. We then summarize our contributions on solving these research problems. Last, we give the organization of the thesis.

1.1 Introduction

Technology advances have tremendously increased the communication and computational powers of many mobile devices, such as laptops, PDAs (Personal Digital Assistants), smart phones, and PMPs (Portable Media Players). These mobile devices, despite their small sizes, have evolved to almost full-fledged mobile computers and can render multimedia content. Therefore, increasingly more users use these mobile devices to watch videos streamed over wireless networks, and they demand more contents at better quality. For example, video broadcast services have been deployed in parts of Europe, Africa, and Asia, and in pilot-testing in several locations in North America and South America [1]. In addition, market forecasts reveal that video streaming, such as mobile TV, will catch up with gaming and music, and become the most popular application on mobile devices: more than 140 million subscribers worldwide, and multi-billion dollar revenues in North America by 2011 [2].

While videos can be delivered using unicast over a cellular network to individual mobile devices, doing so incurs high bandwidth requirement, which grows linearly with the number of mobile devices. Therefore, delivering videos over a cellular network may easily overload

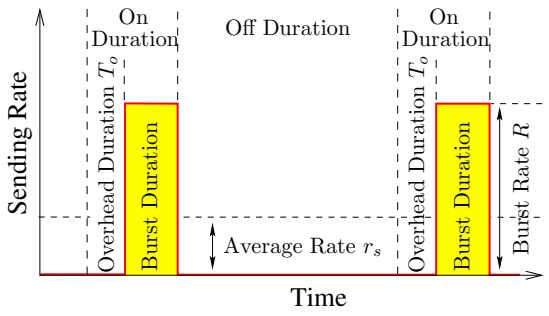


Figure 1.1: Time slicing in mobile broadcast networks to save energy.

the network, and thus does not scale well. Analysis, a market research group, predicted that 3G cellular networks would be overloaded with only 40 percent of cellular phone users watching eight minutes of video per day [3]. In fact, a recent news report reveals that smart phone users are already very close to saturate the bandwidth of 3G cellular networks as of in 2009 [4].

In contrast, streaming videos from a base station using one-to-many multicast/broadcast service achieves high spectrum efficiency, and can support a large number of mobile devices. Therefore, we consider video streaming over wireless networks that support multicast/broadcast, and we call these wireless networks as *broadcast* networks. Sample broadcast networks include WiMAX networks [5, 6], MBMS (Multimedia Broadcast Multicast Services) cellular networks [7], and mobile TV broadcast networks such as DVB-H (Digital Video Broadcast-Handheld) [8–11], MediaFLO (Forward Link Only) [12, 13], and ATSC (Advanced Television Systems Committee) mobile DTV [14] networks. While broadcast networks naturally support live, TV-like, video streaming, they can also support Video-on-Demand (VoD) services using various periodic broadcast techniques. With periodic broadcast, each video is divided into several segments, and each segment is broadcast on multiple broadcast channels at different times [15]. Therefore, periodic broadcast enables mobile devices to quickly find desired segments no matter when they tune to that video. While we do not discuss VoD services in this thesis, most of our solutions can be integrated with techniques such as periodic broadcast to support VoD services.

In broadcast networks, a base station concurrently broadcasts multiple video streams over a shared air medium to many mobile devices. Mobile devices have stringent battery capacity and heat dissipation requirements. Thus, they cannot accommodate additional chips that consume too much energy. For instance, the DVB-H standard [16] states that, with

current battery and semiconductor technologies, signal receivers with power consumption higher than 100 mW cannot be integrated with mobile devices. However, even the state-of-the-art prototype chips consume about 200 mW [17], while commercial chips consume more than 400 mW [18]. Since energy consumption on mobile devices is critical, many broadcast standards, such as DVB-H [8,10,11] and MediaFLO [12], *dictate* using energy saving schemes to increase the viewing time on mobile devices. The typical scheme for saving energy is to make the base station broadcast the video data of a video stream in *bursts* at a bit rate much higher than its encoding rate. Mobile devices, therefore, can receive a burst of traffic and turn off their receiving circuits until the next burst. This is called *time slicing*, and it is illustrated in Figure 1.1 for one video stream. Previous works in the literature show that time slicing is effective in reducing energy consumption on mobile devices [16, 19]. While time slicing enables mobile devices to save energy, it increases the channel switching delay, which is the time that a user waits before s/he starts viewing a selected video stream when a change of video stream is requested by that user. The switching delay is an important performance metric, because many users quickly flip through several streaming videos before they decide on watching the specific ones. Long and variable switching delays are annoying to users and may turn them away from the video broadcast service. Therefore, operators of video broadcast networks have to maintain low and constant switching delays. The energy saving and the channel switching delay are two of the most important Quality-of-Service (QoS) metrics for mobile users.

While network operators must strive to achieve high energy saving and short channel switching delay to retain their subscribers, they also need to maintain high bandwidth efficiency. Bandwidth efficiency refers to the number of video streams that can be concurrently broadcast at a quality no worse than a given target quality within a bandwidth limitation. Bandwidth efficiency is critical to network operators because the wireless spectrum is expensive. For example, AT&T sold a WiMAX spectrum in the southeast USA to Clearwire for $300 million [20], and Inukshuk paid $46 million to license a WiMAX spectrum in Canada [21]. Hence, to be commercially viable, network operators need to achieve high bandwidth efficiency and broadcast as many video streams as possible for higher profits.

In addition to bandwidth efficiency, supporting mobile devices with heterogeneous resources, such as screen resolution, decoder capability, and battery level, is also critical for network operators in order to increase the number of service subscribers. To support heterogeneous devices, network operators may broadcast each video in multiple versions, where

each version is suitable for one mobile device type. This is called *multi-version* video broadcasting. Multi-version broadcasting is inefficient in terms of bandwidth as it effectively reduces the number of *different* videos that can be concurrently broadcast. Hence, network operators must carefully prepare the broadcast signals in order to support heterogeneous mobile devices without sacrificing bandwidth efficiency. The bandwidth efficiency and the support of heterogeneous mobile devices are two of the most important goals for network operators.

In this thesis, we study mobile video broadcast networks in which a base station concurrently transmits multiple video streams over a shared air medium to many mobile devices. We consider three QoS metrics: energy saving and channel switching delay on mobile devices, and bandwidth efficiency in broadcast networks. The goal of this thesis is to optimize broadcast networks from various aspects: we propose several algorithms to (i) maximize energy saving of mobile devices, (ii) maximize bandwidth efficiency of the network, (iii) minimize channel switching delays on mobile devices, and (iv) efficiently support heterogeneous mobile devices. Implementing our algorithms in mobile video broadcast networks provides millions of users better streaming quality and higher user satisfaction, and allows network operators to achieve higher bandwidth efficiency and support more mobile devices for higher revenues.

1.2 Thesis Contributions

In this section, we summarize the research problems studied and addressed in this thesis. We also highlight our contributions for solving each problem.

1.2.1 Energy Saving of Mobile Devices

Time slicing, illustrated in Figure 1.1, is widely used in mobile video broadcast networks to save energy on mobile devices and increase users' viewing times. While time slicing leads to energy conservation, burst schedules, which specify the burst start times and sizes, must be carefully composed to guarantee streaming quality and proper functioning of the system. This is because of a number of reasons. First, since mobile devices have limited receiving buffer capacity, arbitrary burst schedules can result in buffer over/underflow instances that cause playout glitches and degrade viewing experience. Second, as several video streams share the same air medium, burst schedules must not have any burst conflicts, which occur

when two or more bursts intersect with each other in time. Third, turning on receiving circuits to receive next burst is not instantaneous, because it takes receiving circuitry some time to power up and lock on to the radio signals before data can be demodulated [16]. This imposes overhead on energy consumption and this overhead must be considered when constructing burst schedules.

Current practices of computing burst schedules are rather ad-hoc. For example, the heuristic method proposed in the DVB-H standard documents [16, pp. 66] provides schedules for only one video stream. This heuristic simply allocates a new burst only after the data of its preceding burst is consumed by the player at the receiver. This cannot be generalized to multiple video streams with *different* bit rates, because the computed schedule may have burst conflicts and may result in buffer under/overflow instances. Thus, many network deployments had to resort to encoding all video streams at the same bit rate, and simply stagger video streams next to each other. For example, the trial mobile TV service in Paris broadcast 13 video streams all encoded at 270 kbps [22]. Encoding all video streams at the same bit rate is clearly inefficient and may yield huge quality variations among different kinds of video streams. For example, encoding a sports game requires a much higher bit rate than encoding a talk show. If we encode all video streams at the same high bit rate, some video streams may unnecessarily be allocated more bandwidth than they require and this extra bandwidth yields only marginal or no visual quality improvement. Thus, the expensive wireless bandwidth of the broadcast network could be wasted. On the other hand, if we encode all video streams at the same low or moderate bit rate, not all video streams will have good visual quality, which is annoying to users of a *commercial* service.

To the best of our knowledge, there exist no systematic ways in the literature that solve the problem of burst scheduling of video streams coded at different bit rates in order to maximize energy saving on mobile devices, despite the importance of energy consumption on mobile devices. We formally describe the burst scheduling problem to maximize energy saving in the following.

Problem 1 (Maximizing Energy Saving)**.** *Consider several video streams to be concurrently broadcast from a base station to multiple mobile devices. Each video stream is broadcast as bursts of data to save the energy of mobile devices. Construct the optimal burst schedule for all video streams to maximize the system-wide energy saving, while achieving optimal streaming quality, i.e., resulting in no playout glitches on mobile devices. Playout glitches occur when the burst schedule has any burst collisions* or *receiving buffer violations. A burst*

schedule has burst collisions when two or more bursts have nonempty intersection in time. A buffer violation occurs for a video stream when receivers have either no data in the buffer to pass on to video decoders for playout (buffer underflow), or have no space to store data during a burst transmission (buffer overflow).

We address this problem in Chapter 3, and particularly, we make the following contributions [23–26]:

- We formulate a general burst scheduling problem in mobile video broadcast networks. Our formulation is general because each video stream can be coded at any arbitrary encoding rate. The objective of the formulation is to maximize the overall energy saving of all mobile devices. We also show that this problem is NP-complete for video streams with arbitrary bit rates [23].

- Since the general problem may not be efficiently solved, we first propose a practical simplification of the general problem, which allows video streams to be classified into multiple classes and each class has a different bit rate. The bit rate of class c, r_c, can take any value in the form of $r_c = 2^i \times r_1$, where $i \in \{0, 1, 2, 3, \dots\}$, and r_1 is the bit rate of the lowest class. r_1 can take any arbitrary bit rate. Using this simplification, we develop an optimal (in terms of energy consumption) burst scheduling algorithm. We analytically show that the proposed algorithm is efficient [24].

- We implement our algorithm for the simplified problem in a mobile TV testbed and demonstrate its practicality and effectiveness in saving energy. We also conduct extensive simulations to quantify the performance of our algorithm under wide ranges of broadcast parameters.

- We solve the general formulation in which each video stream can be coded at any arbitrary encoding rate. Because the problem is NP-complete and cannot be optimally solved within reasonable amount of time, we propose an approximation algorithm for it. We prove the correctness of our proposed algorithm, and we show that it runs in polynomial time. Moreover, we analytically and numerically show that burst schedules produced by the proposed algorithm achieve near-optimal energy saving [25, 26].

- We implement our approximation algorithm in a mobile TV testbed to show its practicality and efficiency. We also implement this algorithm in a simulator, which captures

important aspects related to the burst scheduling problem but abstracts away irrelevant details. We use this simulator to exercise the proposed algorithm with wider ranges of parameters that are difficult to set in the real testbed.

1.2.2 Bandwidth Efficiency of Broadcast Networks

While solving Problem 1 allows network operators to provide mobile streaming services with high energy saving and optimal streaming quality, it does not consider the bandwidth efficiency of broadcast networks. Bandwidth efficiency is critical for network operators because higher bandwidth efficiency generally leads to more concurrent video streams within a given network bandwidth. One way to increase bandwidth efficiency is to encode videos in VBR (Variable-Bit-Rate), rather than CBR (Constant-Bit-Rate), because VBR coding achieves higher coding efficiency and higher statistical multiplexing gain [27]. Coding efficiency refers to the ratio between video quality and encoding bit rate, while statistical multiplexing gain refers to the increase in number of video streams that can be concurrently broadcast at a target video quality using a given network bandwidth. More details about VBR coding are given in Section 2.2.1.

The higher statistical multiplexing gain of VBR streams is due to the fact that each video contains scenes with different scene complexity, and multiple videos are unlikely to have high complexity scenes at the same time. Therefore, by adjusting the encoding rates of individual videos based on their scene complexities, network operators may better utilize the bandwidth of their broadcast networks. Packet switched networks, such as the Internet, achieve statistical multiplexing by dividing data into small packets (e.g., up to 1.5 KB in the Internet [28]) and then routing each packet separately. However, doing so in mobile video broadcast networks may result in low energy saving and/or playout glitches, as these networks usually transmit data in much larger bursts: in the order of a few hundreds KB [16]. Therefore, base stations must carefully construct bursts for multiple VBR video streams to achieve optimal streaming quality and high energy saving on mobile devices, while maximizing the *goodput* in the broadcast network. The goodput refers to the fraction of the amount of ontime delivered video data over the broadcast network capacity. Goodput includes only the video data delivered *before* their decoding deadlines, as late video data cannot be rendered to users and are essentially useless.

To the best of our knowledge, optimally broadcasting multiple VBR streams over wireless networks has not been fully addressed in the literature. Recent papers [29, 30] emphasize

that efficiently broadcasting VBR streams is one of the most critical and challenging *open* problems in mobile broadcast networks. We formally describe the burst scheduling problem of VBR streams to maximize network goodput and energy saving in the squeal.

Problem 2 (Maximizing Goodput)**.** *Consider several VBR video streams to be concurrently broadcast by a base station to multiple mobile devices. Each video stream is broadcast as bursts of data to save energy on mobile devices. Construct the optimal burst schedule for all video streams to maximize the goodput of the broadcast network* and *the energy saving on mobile devices, while achieving optimal streaming quality, i.e., resulting in no playout glitches on mobile devices.*

We solve this problem in Chapter 4. In particular, we make the following contributions [31]:

- We formulate the problem of broadcasting multiple VBR streams from a base station to many mobile devices, in order to maximize: (i) the goodput in the network, (ii) the energy saving of mobile devices, and (iii) the streaming quality on mobile devices. We show that this problem is NP-complete [31].
- We consider our problem in 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 buffer violations and playout glitches.
- 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 [31]. 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 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 broadcast networks.

- We also 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 good streaming quality, high goodput, and high energy saving.

- Although the proposed scheduling algorithm works in open-loop broadcast networks, it may result in some playout glitches when the total bit rate of all video streams exceeds the network bandwidth. To address this issue, we propose a new scheduling algorithm that employs longer lookahead windows and utilizes slack times of the air medium for fewer playout glitches.

1.2.3 Channel Switching Delay of Mobile Devices

Every network operator concurrently broadcasts multiple video streams over a broadcast network, and each user tunes his/her mobile device to receive a single video stream at any time. That is, users of mobile broadcast services often switch among many video streams, or better known as *channels*[1], before they decide on watching the specific ones. Switching video channels incurs channel switching delays. Long and variable channel switching delays are annoying to users and may turn them away from the mobile video broadcast service. For example, the Digital Television consortium of Northern Europe recommends a maximum switching delay of 1.5 sec for digital TV services [32]. Therefore, in addition to video streaming quality and energy saving, network operators must also maintain low and constant switching delays, in order to achieve good user experience.

While time slicing saves energy on mobile devices, it also increases channel switching delay because the video data are broadcast in bursts. More precisely, channel switching delay is composed of several parts, in which *frame refresh delay* and *time slicing delay* are the two dominating contributors [29, 33]. The frame refresh delay refers to the time period between receiving the first bit of a new video stream and receiving the next random access point, typically an intra-coded frame, of that video. Frame refresh delay is controlled by video coders in the application layer, and thus is orthogonal to burst scheduling in the link layer. The time slicing delay refers to the time period between locking onto a channel and reaching the first burst of that channel. Since time slicing delay is a *by-product* of the

[1] We interchangeably use video streams and channels throughout this thesis.

burst scheduling algorithms, we only consider time slicing delay throughout this thesis, and assume all other components of the channel switching delay are fixed.

We consider the problem of controlling the switching delay in broadcast networks that employ time slicing to save energy. Our goal is to provide a guarantee on the maximum switching delay from a TV channel to any other channel *without* sacrificing the energy saving of mobile devices. We formally describe the considered problem in the following.

Problem 3 (Bounding Channel Switching Delay)**.** *Consider several video streams to be concurrently broadcast by a base station to multiple mobile devices. Construct the optimal burst schedule for all video streams to guarantee that the channel switching delay from any stream to any other stream does not exceed a maximum allowable channel switching delay, while maximizing the energy saving on mobile devices and achieving optimal streaming quality, i.e., resulting in no playout glitches on mobile devices.*

We address this problem in Chapter 5, and particularly, we make the following contributions [34, 35]:

- We analyze the time slicing scheme currently used in many deployed mobile video broadcast networks, and we show that it is not efficient in terms of energy saving of mobile devices, especially when short channel switching delays are required [34].
- We propose three new time slicing schemes that ensure that a given maximum switching delay is not exceeded, while at the same time the energy saving of mobile devices is maximized. The new time slicing schemes employ simulcasting of video streams with and without scalable video coding techniques (described in Section 2.2.2) [35].
- We prove the correctness of the proposed schemes and derive closed-form equations for the achieved energy saving. We numerically analyze the performance of the proposed schemes and provide guidelines on choosing the most suitable time slicing scheme for a given mobile video broadcast network.
- We implement the proposed schemes in a mobile TV testbed. Our experimental results validate our theoretical analysis and show that the proposed schemes indeed meet the target channel switching delays and achieve high energy savings for mobile devices.

1.2.4 Supporting Heterogeneous Mobile Devices

In the previous problems, we assume mobile devices are homogeneous and we broadcast each video stream once for all mobile devices. In reality, mobile devices have heterogeneous resources such as screen resolution, decoder capability, and battery capacity. For example, while laptop computers can display 720p (1280x720) videos, most smart phones only have QVGA (320x240) displays. Therefore, it is important to concurrently support all these mobile devices: broadcasting a video stream in QVGA resolution results in unacceptable video quality on laptop computers, while broadcasting in high resolution results in higher overhead, and thus higher energy consumption, on smart phones with no visible quality improvement. More importantly, broadcasting in 720p resolution could deny smart phones that are not computationally powerful enough from the mobile video broadcast services. To partially cope with this problem, network operators could broadcast every video in multiple versions, where each version targets a type of mobile device. This is known as multi-version video broadcasting. Multi-version broadcasting is inefficient in terms of bandwidth as it effectively reduces the number of video streams that can be concurrently broadcast.

Scalable video broadcasting, in contrast, enables network operators to support various mobile devices without exhausting network bandwidth. This is achieved by using scalable video coders to encode each video into a single stream with multiple layers, and broadcast each layer only once. Such a coded stream is scalable because several substreams, with one or a few layers, can be extracted from the complete stream and are still decodable. Each mobile device can then choose and render the substream that is most appropriate to its capability and network conditions. Broadcasting scalable video streams in broadcast networks that employs time slicing to save energy is difficult, because the base station must prepare burst schedules to achieve optimal streaming quality while carefully considering the dependency among layers.

We study the scalable video broadcasting problem in mobile video broadcast networks, where each video is encoded into a scalable video stream with multiple layers, and several video streams are concurrently broadcast over a shared air medium to many mobile devices with heterogeneous resources. We formally describe our problem in the squeal.

Problem 4 (Supporting Heterogeneous Devices)**.** *Consider several scalably-coded video streams to be broadcast by a base station to heterogeneous mobile devices, which have diverse capability and can decode different substreams extracted from the complete scalable*

video streams. Organize the scalable video streams into data bursts and construct a burst schedule for all video streams for each mobile device to efficiently receive the most appropriate substream while achieving high energy saving and low channel switching delay from any channel to any other channel. The appropriate stream of each mobile device may depend on the device capability, the target energy consumption level, and user preferences.

We solve this problem in Chapter 6. In particular, we make the following contributions [36, 37]:

- We analyze the current mobile video broadcast networks and we show that they are not efficient for scalable video streams. This is done by first presenting several time slicing schemes to enable scalable video broadcasting in current systems, and then pointing out their drawbacks. Moreover, we analytically show that these time slicing schemes lead to lower energy saving compared to our proposed schemes [36].
- We design two time slicing schemes to support heterogeneous devices using scalable video streams [37]. The scalable video streams may have different layer bit rates, which allow the coded streams to be better matched with the capability of various types of mobile devices. We analytically prove that both schemes achieve high energy saving. In addition, one of the schemes is designed to also maintain low channel switching delays.
- We implement these two time slicing schemes in a real mobile TV testbed to demonstrate their practicality and efficiency. The experimental results indicate that the proposed schemes allow mobile devices to trade perceived quality for energy saving, as they can opt to receive a smaller substream to prolong battery lifetime.

1.3 Thesis Organization

The rest of this thesis is organized as follows. We present some background about mobile video broadcast networks as well as video coding techniques in Chapter 2. We formulate and solve the problem of maximizing energy saving in Chapter 3. In Chapter 4, we consider the problem of maximizing goodput and energy saving using VBR video streams. We consider the problem of controlling channel switching delays in Chapter 5, and we present time slicing

schemes to support heterogeneous mobile devices in Chapter 6. We conclude the thesis and outline future research directions in Chapter 7.

Chapter 2

Background

In this chapter, we provide a background about mobile video broadcast networks and video coding techniques. We also present rate regulation techniques that allow network operators to cope with the bit rate variations of VBR streams. We describe the burst scheduling algorithms currently used in commercial broadcast base stations, and we empirically measure the actual energy consumption of real mobile devices.

2.1 Broadcast Network Standards

Delivering video streams to mobile devices can be done over wireless cellular networks, generic packet switched networks, and over dedicated broadcast networks. We survey each of them below. We focus more on dedicated broadcast networks as they are the main target of our research.

2.1.1 Cellular and MBMS Networks

Despite the broadcast nature of wireless communication, traditional cellular networks only support unicast, and thus are not bandwidth efficient especially in urban areas where there are many users receiving the same content. More specifically, with unicast, the base station transmit a copy of video stream to each mobile device, which can easily saturate the network bandwidth. To cope with this problem, the 3G Partnership Project (3GPP) has defined an integrated multicast and broadcast extension, called MBMS [38], for Universal Mobile Telecommunications Systems (UMTS). MBMS allows many mobile devices, within the range

of a cellular tower, to receive the same video stream without duplicating video streams in wireless and wire networks, and thus become more bandwidth efficient.

2.1.2 WiMAX Networks

IEEE 802.16 WiMAX [6, 39] is a standard for metropolitan area wireless networks. WiMax is envisioned as an alternative solution to traditional wire-based broadband access. For the emerging countries like China, Russia and India, WiMAX is a cost-effective last-mile solution, and they are expected to be the major WiMAX market. WiMax has the capability of delivering high-speed services up to a range of 30 miles. WiMax uses Orthogonal Frequency Division Multiplex (OFDM) and Orthogonal Frequency Division Multiple Access (OFDMA) to improve the transmission range and increase bandwidth utilization. OFDM prevents inter-channel interference among adjacent wireless channels, which allows WiMAX to achieve high network bandwidth. In link layer, WiMAX supports QoS differentiation and ensures the target bandwidth and latency of each QoS class are met. Multicast and broadcast services are also supported in WiMAX. Therefore, WiMAX is suitable for broadcasting video streams that impose high data volume and stringent QoS requirements. In most common WiMAX networks, the wireless channel is divided using time division into frames. Each frame is divided into downlink subframe and uplink subframe. The downlink subframe is used by the base station to broadcast to all wireless stations. The uplink subframe is further divided into variable-length transmission periods, where each period is allocated to a wireless station to transmit data to the base station. The length of each transmission period is computed by the base station and is included in the beginning of each frame.

2.1.3 Dedicated Video Broadcast Networks

Overview

There are several standards for dedicated video broadcast networks, including T-DMB (Terrestrial-Digital Multimedia Broadcasting) [40], ISDB-T (Integrated Services Digital Broadcasting-Terrestrial) [41], MediaFLO [12, 13], and DVB-H [8–11].

A brief overview of each follows. T-DMB [40] is an extension for the DAB (Digital Audio Broadcast) standard [42] to add video broadcast services to the high-quality audio

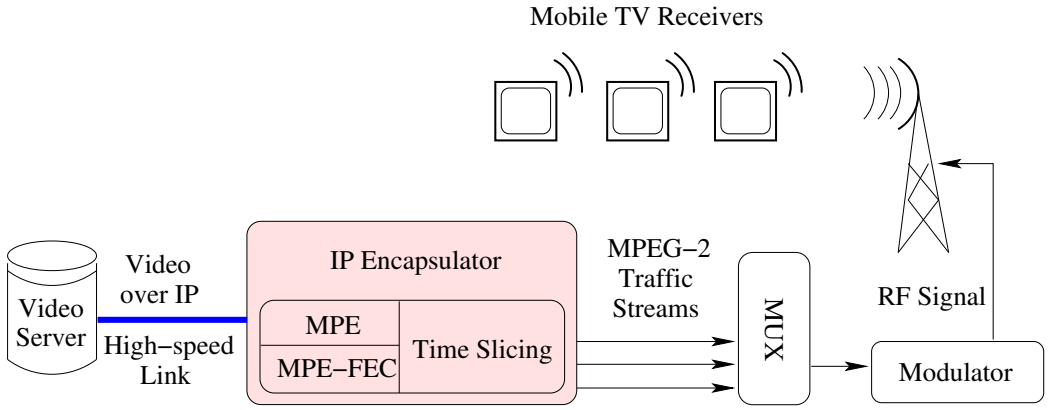


Figure 2.1: The main components of mobile TV broadcasting systems.

services offered by DAB. The extension includes both source coding, such as using MPEG-4/AVC encoding, and channel coding, such as employing Reed-Solomon (R-S) code. The development of T-DMB is supported by the South Korean government, and T-DMB is the first commercial mobile video broadcast service. In addition to South Korea, several European countries may deploy T-DMB as they already have equipment and experience with DAB systems. ISDB-T [41] is a digital video broadcast standard defined in Japan, which is not only for fixed video receivers but also for mobile receivers. ISDB-T divides its spectrum into 13 segments, where 12 of them are used for broadcasting HDTV and one is for broadcasting to mobile devices. T-DMB and ISDB-T are narrow bandwidth networks and cannot employ time slicing technique to save energy on mobile devices. In contrast, MediaFLO and DVB-H networks have much higher bandwidth and can save energy using time slicing [12, 43].

MediaFLO Networks

MediaFLO [12] is a video broadcast network developed by Qualcomm and the FLO forum [44]. MediaFLO is designed from scratch for video broadcast services to mobile devices. The details of the design are not public. In contrast, DVB-H [9, 10] is an open international standard [8]. We use the open DVB-H standard in our discussion throughout the thesis. Nonetheless, in our problem formulation and solution we abstract away the specific details of the DVB-H standard. Therefore, our solution is also applicable to the MediaFLO system and other wide band video broadcast networks that may be developed in the future.

DVB-H Networks

With participants from over 35 countries, the Digital Video Broadcast (DVB) consortium started designing a series of international standards to support digital television and data broadcast services in 1993. Many of these standards, such as DVB-S for broadcasting over satellite links, DVB-C for broadcasting over cables, DVB-T for broadcasting over airwaves, have been widely deployed. It is reported that more than 170 million DVB receivers have been sold at the time of writing [45]. Several updates of these standards, the second-generations, are being actively developed. In the late 90's, DVB consortium investigated the potential of receiving DVB-T signals using mobile devices. They have concluded that with a spatial diversity antennas for better reception, DVB-T signals can be received by mobile devices [46].

While DVB-T can support some mobile usages, it is not suitable for streaming multimedia contents to small mobile devices such as cellular phones for three reasons. First, most mobile devices are battery-powered and have very limited power. Unfortunately, power consumption was never a considered factor when designing DVB-T standard as DVB-T receivers are often powered by external sources. Second, mobile devices suffer from serious and varying Doppler shift, fading and interference because their various degree of movements and heterogeneous environments (in-door, out-door, or in-car). In addition, many of these devices concurrently support broadcast and cellular networks, which leads to more inter-system interference. This problem is even more severe considering these mobile devices can only adopt small built-in antennas, which rules out most of the advanced antennas such as direction and spatial diversity antennas, and results in poor antenna gain. Thus, radio performance better than what can be achieved by the DVB-T standard is desired for robust broadcasting. Third, mobile devices require a faster, soft, handoff mechanism as they are often on-the-move. To cope with these challenges, the DVB consortium developed the DVB-H standard [8], which is an extension of the DVB-T standard but tailored to mobile devices. DVB-H, published in 2004, addresses the three requirements mentioned above. Currently, trial or full-service DVB-H networks have been deployed in many countries [1]. Figure 2.1 illustrates the main components of a DVB-H system.

The DVB-H standard defines protocols below the network layer and uses IP as the interface with the higher-layer protocols as illustrated in Figure 2.2. The DVB-H standard uses a physical layer compatible with the DVB-T standard, which employs Orthogonal Frequency

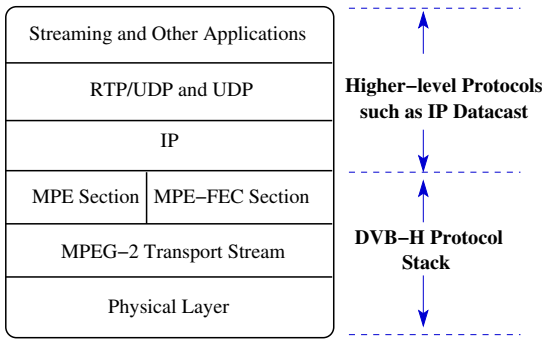


Figure 2.2: The protocol stack of mobile TV networks using the DVB-H standard.

Division Multiplexing (OFDM) modulation. To support interactive end-to-end broadcasting systems, the DVB consortium started the development of the IP Datacast standard in 2004, which not only specifies higher layer protocols but also enables cooperation with cellular networks such as UMTS. Incorporating access to cellular networks provides bidirectional communications and enables many interactive services such as Electronic Service Guide (ESG). The IP Datacast standard was finalized in 2007 [47].

DVB-H encapsulates IP packets using Multi-Protocol Encapsulation (MPE) sections to form MPEG-2 transport streams (TS). Thus, data from a specific TV channel form a sequence of MPEs. The MPE encapsulation is done in a module called the *IP encapsulator*. The encapsulated data is then fed to an RF signal modulator to be transmitted over the air medium. The IP encapsulator realizes two additional features of DVB-H: time slicing and forward error correction (FEC). In the following, we describe these two features, which provide several performance optimization opportunities.

To save energy of mobile devices, MPEs belonging to a given TV channel are transmitted in *bursts* with a bit rate much higher than the video stream itself. Thus, mobile devices can receive a burst of traffic and then turn off their receiving circuits till the next burst. This is known as time slicing. The time period between two adjacent bursts (the off period) is flexible in the sense that the time offset between the start time of the next burst and the current MPE section is sent as part of the MPE header. This enables DVB-H systems to adopt variable burst durations and off durations not only for different video streams but also for the same video stream at different times. We note that the activation of the receiving circuits is not instantaneous for two reasons: delay jitter and channel synchronization. Since the next burst start time is broadcast as an offset to the current time, constant transmission

delay does not affect its accuracy. However, delay jitter skews the start time of the next burst. This jitter is accommodated by adding an extra jitter delay after each off duration, which prevents collisions among bursts. In addition, synchronization time refers to the time for receiving circuits to search for and lock on the broadcast frequency. We collectively refer to the sum of jitter delay and synchronization time as *overhead duration* since no data is transmitted during this period. We use T_o sec to denote the overhead duration, which is a system parameter of broadcast networks. In addition to energy saving, with time slicing, mobile devices can use their receiving circuits to search for signal in adjacent cells between two bursts for seamless, soft, handoffs. Handoffs happen when a mobile device moves from a broadcast cell to an adjacent cell. Soft handoff means a device locks up to the signal in the new cell before ignoring the signal in the old cell; hence, there is no service interruption during soft handoffs, which leads to good user-experience. Without time slicing, auxiliary receiving circuits are required to support soft handoffs, which lead to higher costs.

The DVB-H standard applies R-S code and time interleaving in its link layer to protect IP packets in each burst transmission, which largely reduces the minimal carrier-to-noise (C/N) ratio for successful decoding with the same antenna gains. This C/N improvement is reported to be equivalent to the antenna gain given by the spatial diversity [8], however, no additional space is taken nor more power is consumed by the spatial-diversity antenna module. To compute the R-S parity bits, DVB-H uses an MPE-FEC frame with 255 columns and at most 1024 rows, where each element is a byte. Therefore, the maximum size of MPE-FEC frame is about 255 KB. Each MPE-FEC frame is divided into an application data table (ADT) with 191 columns of IP packets and an R-S data table (RSDT) with 64 columns of parity bits, where the parity bits are calculated row-wise from the IP packets in the same MPE-FEC frame. Once the R-S bits are ready, IP packets in the same MPE-FEC frame are sequentially transmitted as MPE sections, which are followed by the R-S bits encapsulated in multiprotocol encapsulation sections. At the receiver side, CRC-32 section trailers and R-S parity bits are used for error detection and correction from transmission errors. Notice that, accessing MPE-FEC frames is column-wise while calculating parity-bit is row-wise, which leads to time interleaving that also helps to combat fading and interference. In common cases, default R-S coding ratio supports recovery from as high as 25% packet loss ratio [9]. However, DVB-H supports adaptive R-S coding ratios: padding all zero packets in ADT leads to stronger R-S coding, while puncturing some parity bits results in weaker R-S coding. When MPE-FEC is jointly used with time slicing, each transmission burst is

correspondent to a MPE-FEC frame. As a consequence, the burst size cannot go beyond 255 KB, which simplifies the receiver hardware design as a fixed memory can be reserved for decoding.

While DVB-H employs MPEG-2 transport streams (TSes) to carry data, the video streams are not directly put in TSes, but in the IP packets that are in turn encapsulated in TSes. Since both video stream data and singling messages are IP-based, many IETF-defined protocols are adopted by IP Datacast standard [10, 11] for a complete mobile TV service. Above the network layer, IP Datacast uses UDP protocol in its transport layer. There are two protocols on top of the UDP protocol for different content delivery: real-time streaming protocol (RTP) for multimedia streaming traffic and file delivery over unidirectional transport protocol (FLUTE) for file and meta-data transfers. While IP Datacast chooses H.264/AVC and VC-1 (Windows Media format) for video coding and MPEG4 AAC+ for audio coding, any codec that supports streaming over IP networks can be used. IP Datacast also defines the XML-based Electronic Service Guide (ESG) that delivers TV channel information to users and TV channel initialization parameters to mobile TV receivers. More precisely, ESG uses Session Description Protocol (SDP) to describe initialization parameters. SDP is extensible, e.g., it has been extended to support H.264/SVC scalable coders [48].

2.2 Video Coding Standards

2.2.1 CBR and VBR Coding

Video coders can be roughly categorized into constant bit rate (CBR) and variable bit rate (VBR) coders. CBR coders adjust the coding parameters to maintain a fixed frame size, and thus a constant bit rate throughout a video. Algorithms that analyze video complexity and determine coding parameters for individual frames are called rate control algorithms. CBR coded streams have fixed bandwidth requirements, which largely simplify the problem of bandwidth allocation in packet switched networks. Encoding a video in CBR requires us to either over-subscribe the network bandwidth in best-effort networks, or make a bandwidth reservation in reservation based networks.

While streaming videos coded in CBR is less complicated, doing so leads to degraded user experience and lower bandwidth efficiency. This is because each video consists of scenes with diverse complexities, and encoding all scenes at the same bit rate results in quality fluctuations which are annoying to users. Moreover, to maintain a minimum target quality,

a video must be encoded at a bit rate high enough for the most complex scene to achieve that target quality [27]. This in turn leads to wasting of bandwidth, because less complex scenes are unnecessarily coded at high quality that has little or no impact on human perception.

To address these two issues, VBR coders can be used to provide constant video quality and avoid wasting of bandwidth. This is achieved by dynamically distributing the available bandwidth among frames of the same video so that more complex scenes and more critical frames get more bits. That is, VBR coders enable bit budget redistribution along the time-axis of a given video, and achieve better coding and bandwidth efficiency. VBR coders that encode videos without considering buffer and network status is called unconstrained VBR (UVBR) coders. Since the individual frame sizes are not constrained by any constraints, UVBR coders encode videos only from the perspective of coding efficiency, and achieve the best possible video quality. UVBR streams, however, consist of high bit rate variability, and the variability is increasingly significant in modern video coders such as H.264/AVC [49]. Therefore, smoothly streaming UVBR streams coded by modern coders becomes very challenging, because the coded streams may require instantaneous bit rates much higher than the bandwidth of the underlaying networks.

Constrained VBR (CVBR) coders take one or more target streaming environments as input, and create a VBR stream that can be smoothly streamed under the target streaming environments. Common streaming environments include channel bandwidth, smoothing buffer size, and initial buffering delay [50]. To ensure smooth playouts, CVBR coders implement rate control algorithms, which monitor the complexities of scenes and frames, determine the target size of each frame, and constrain the frame size while encoding a video. The rate control algorithms are similar to those used in CBR coders, but in CVBR coders bit rate variability is allowed as long as it is within the constraints imposed by the target streaming environments.

To prepare coded streams for packet switched networks, service providers may choose different video coders for various reasons. Service providers who would trade bandwidth efficiency for deployment simplicity may choose CBR coders, and deploy a reservation based network, such as WiMAX, or over-subscribe bandwidth, such as deploying an isolated Gigabit Ethernet for each residential user. Service providers which care about bandwidth utilization may prefer VBR coders for the higher coding efficiency. When distribution networks support bandwidth reservation, or the target streaming environments are known at the encoding time, the service providers may choose CVBR coders for fewer playout glitches

and thus better streaming quality. Finally, service providers may choose UVBR coders if the streaming conditions are not known at the encoding time.

Statistical Multiplexing

Statistical multiplexing refers to the capability of sharing the network bandwidth among multiple data streams that have variable bit rate requirements. Statistical multiplexing allows the network to achieve higher bandwidth efficiency. Packet switched networks achieve statistical multiplexing by dividing each data stream into small packets and routing each packet independently over potentially diverse paths to the destination. This allows routers to interleave packets of different data streams on network links in order to share the link bandwidth among data streams. More specifically, packets arrive at each router are first stored. The router then runs a packet scheduling algorithm to determine the next packet to be transmitted. In general, using VBR streams enables network operators to multiplex more videos over a bandwidth limited network, because the bit rate of each video is proportional to its current scene complexity, and not many VBR streams would require high bit rate at the same time. Sharing the network bandwidth among multiple VBR streams achieves statistical multiplexing, and the increase in number of video streams that can be broadcast at a given target video quality by VBR coding is called statistical multiplexing gain.

2.2.2 Scalable Video Coding

Traditional video coders compress each video sequence into a coded stream at a user-specified spatial resolution, temporal frequency, and fidelity level. These video coders are called *non-scalable* video coders because a decoder can either decode a coded stream at its full-quality as specified at encoding time, or cannot produce any meaningful reconstruction. Nonscalable coders are not suitable for modern video communication systems that are in general built over dynamic RTP/IP networks and various end systems with different decoding capabilities [51]. Scalable video coders, in contrast, encode a video sequence into a base layer that provides basic video quality and one or a few enhancement layers that add incremental quality refinements. These layers are then encapsulated into a single scalable stream that can be decoded at various qualities. At the receiver side, a scalable video decoder extracts enhancement layer bits based on its capabilities as well as current network conditions and reconstructs a video sequence at proportional quality. Compared to concurrently streaming

several nonscalable coded streams at different bit rates, streaming a scalable coded stream simplifies the transmission and synchronization issues and saves bandwidth consumption. For example, in H.264/SVC [51], switching from one medium grained scalable (MGS) layer to another can be done at any time, while switching from one nonscalable stream to another can only be done at random access points, which typical are intra-coded frames.

In the past two decades, scalable video coding has been extensively studied in the literature. In addition, several standardization efforts, such as MPEG-2, H.263, and MPEG-4 Visual, have defined some coding tools to support scalable video coding. However, these scalable coding tools are seldom used in actual applications because of coding efficiency and decoding complexity. More specifically, a scalable coded stream results in lower quality compared to a nonscalable coded stream when both streams are decoded at the same bit rate. Moreover, a scalable coded stream incurs additional decoding operations at the decoder side that increases the decoding complexity. Since 2003, these two shortcomings of previous scalable coding standards have stimulated the development of H.264/SVC [52] as an scalable extension to the state-of-art nonscalable coder H.264/AVC [53]. H.264/SVC has been recently finalized by the ITU-T VCEG group and MPEG video group and has become an annex of H.264/AVC. H.264/SVC supports several types of scalability, including display resolution (spatial) scalability, frame rate (temporal) scalability, fidelity (quality) scalability and hybrid scalability of some or all of the above. More importantly, H.264/SVC achieves comparable coding efficiency to nonscalable coders without incurring too much decoding overhead [51]. High coding efficiency and low decoding complexity make H.264/SVC very competitive in real-life applications, including wireless video streaming [54] and Google Video Chat [55], H.264/SVC has become a focus of many codec chip manufactures. For example, only a few months after H.264/SVC standard is finalized, Stretch announced their decoder solution for mobile devices in Spring 2008 [56]. More H.264/SVC decoder chips are expected to hit the market soon.

2.3 Mobile Video Broadcast Systems

A mobile video broadcast system is illustrated in Figure 2.3, which consists of three entities: content providers, network operators, and mobile users. Content providers are companies that create videos. Since content providers send the same video to multiple network providers that have diverse needs, the newly created videos are encoded in high-quality

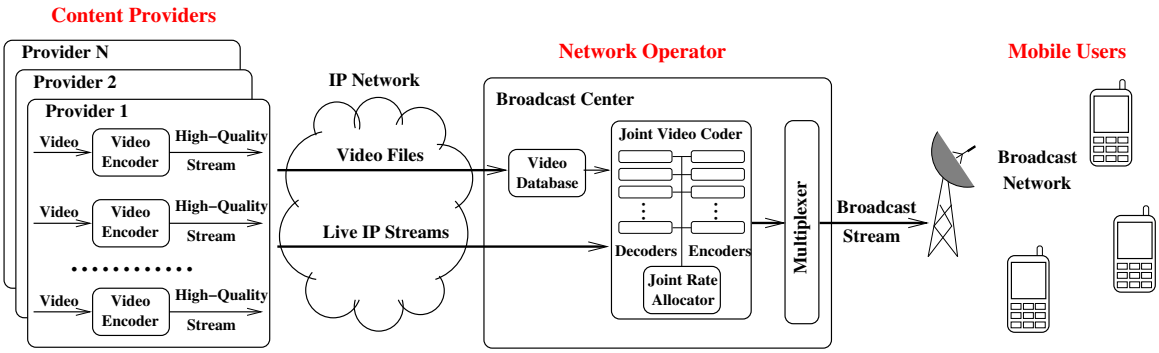


Figure 2.3: Main components of a closed-loop video broadcast system.

streams and sent to network operators. Each network operator may transcode the video stream following its quality requirements and available bandwidth. 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. 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 2.3, a joint video coder consists of a joint rate allocator, several decoders, and several VBR coders. The joint rate allocator collects scene complexities from decoders/coders, distributes available bandwidth among video streams, and instructs the coders to avoid overloading the broadcast network by controlling their encoding bit rates. There are several joint video coders proposed in the literature, e.g., [29, 57–59]. Commercial joint coders, such as [60], are also available in the market. We call broadcast systems with joint video coders as *closed-loop* broadcast systems. While deploying joint video coders can simplify the design of the multiplexer, doing so may not always be possible for several reasons, such as complex business agreement and higher deployment cost. Furthermore, setting up closed-loop broadcast systems for temporary services, such as systems for special sports events and emergency services, leads to high configuration overhead and thus may not be possible. We call broadcast systems with standalone video coders as *open-loop* broadcast systems. We

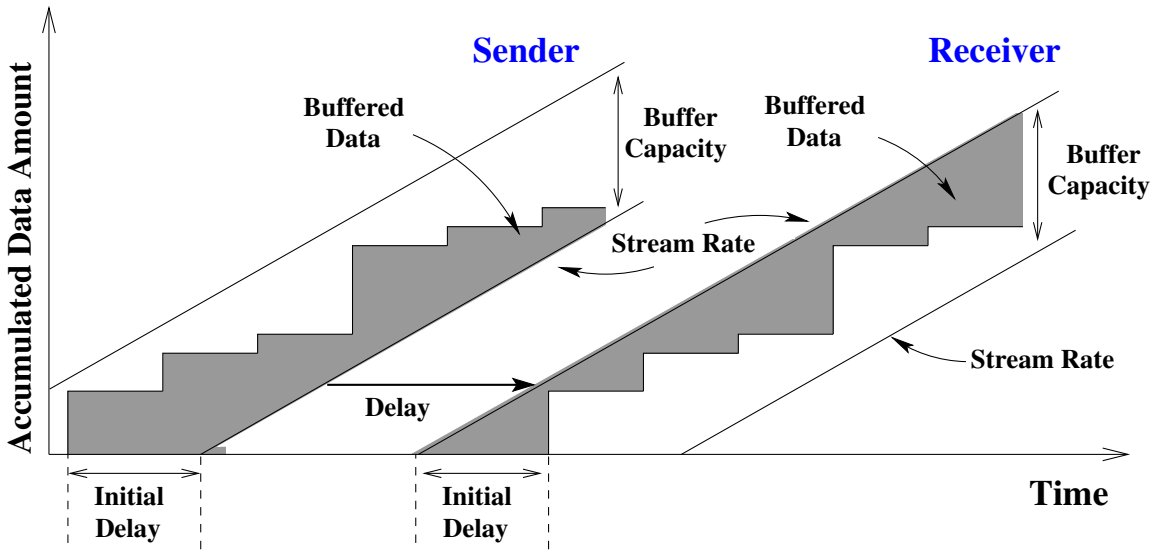


Figure 2.4: Dynamics of the rate regulation buffers of sender and receiver.

notice that video broadcast systems that broadcast CBR video streams are essentially closed-loop broadcast systems, because the aggregate bit rate of all video streams is constant in these systems. In either closed- or open-loop networks, the multiplexer generates a broadcast stream, which in turn is broadcast via a tower to mobile users. Mobile users use mobile devices to receive video streams over the broadcast network. Each user tunes to a video stream at any moment.

2.4 Rate Regulation of VBR Streams

As mentioned in Section 2.2.1, CBR coded streams incur coding inefficiency and high quality fluctuation, and are not suitable to mobile video broadcast systems. Modern video coding standards, such as H.264/AVC [61], support CBR coding in a relaxed sense, where streaming servers employ a rate regulation buffer to *regulate* VBR coded streams into CBR streams at streaming time. We describe the rate regulation process in the following.

The rate regulation operation can be described by the Hypothetical Reference Decoder (HRD) model of H.264/AVC [62], which guides streaming systems to properly set up buffer to smooth out the streaming traffic without introducing any buffer under/overflow instances for smooth playouts. Similar to hypothetical reference decoders defined in previous video

coding standards, the H.264/AVC HRD model is based on the leaky bucket model. Figure 2.4 illustrates the buffer dynamics at both the sender (on the left) and the receiver (on the right), which are connected by a CBR communication channel with a constant delay. At the sender side, each coded frame is instantaneously inserted into the sender's buffer, where the coded frames can be generated by an online video coder or read from a file that was generated offline. Since coded frames can have different sizes, the accumulated data amount of the video stream is a staircase, where the heights of steps are different. The buffered data is *drained* and sent to the receiver at a *constant* bit rate r, where r is the slope of the straight line beneath the staircase, and the area below the staircase (shaded in the figure) represents the remaining buffered data in the sender's buffer.

After a constant transmission delay, the data arrives at the receiver also at bit rate r, and is stored in the receiver's buffer. Since coded frames have various sizes, the video decoder at the receiver waits for enough number of bits, and instantaneously removes each coded frame from the receiver's buffer for decoding, which is represented as another staircase in this figure. The area above this staircase (shaded in the figure) represents the remaining buffered data in the receiver's buffer. We note that the upper-left line at the sender side indicates the sender's buffer limit, and the lower-right line at the receiver side indicates the receiver's buffer limit.

Note that no buffer under/overflow is possible if the staircase (at either sender or receiver) stays within the tube of the two straight lines, and the tube can be uniquely specified by its streaming rate, buffer capacity, and initial delay [62]. Therefore, streaming systems, including mobile video broadcast systems, can employ the HRD model and use rate regulation buffers to regulate the streaming rates of video streams, although the frames were not coded in uniform size. We refer to VBR streams that are rate regulated as *rate regulated* VBR streams, which is a type of CVBR streams defined in Section 2.2.1.

Rate regulated VBR streams can be broadcast in mobile video broadcast systems as illustrated in Figure 2.5. We use some mobile TV terminology, defined in Section 2.1.3, in the following description. At the base station, from the left to right, the coded frames are inserted into a Rate Regulation Buffer that adopts a traffic regulator to produce a CBR stream at a constant streaming rate. This CBR stream is sent over RTP packets to the Multi-Protocol Encapsulation Buffer, where they form MPE bursts, which are larger than the RTP packets. MPE bursts are then broadcast over the air medium, and stored in Multi-Protocol Decapsulation Buffer before being decapsulated back to a CBR stream at the

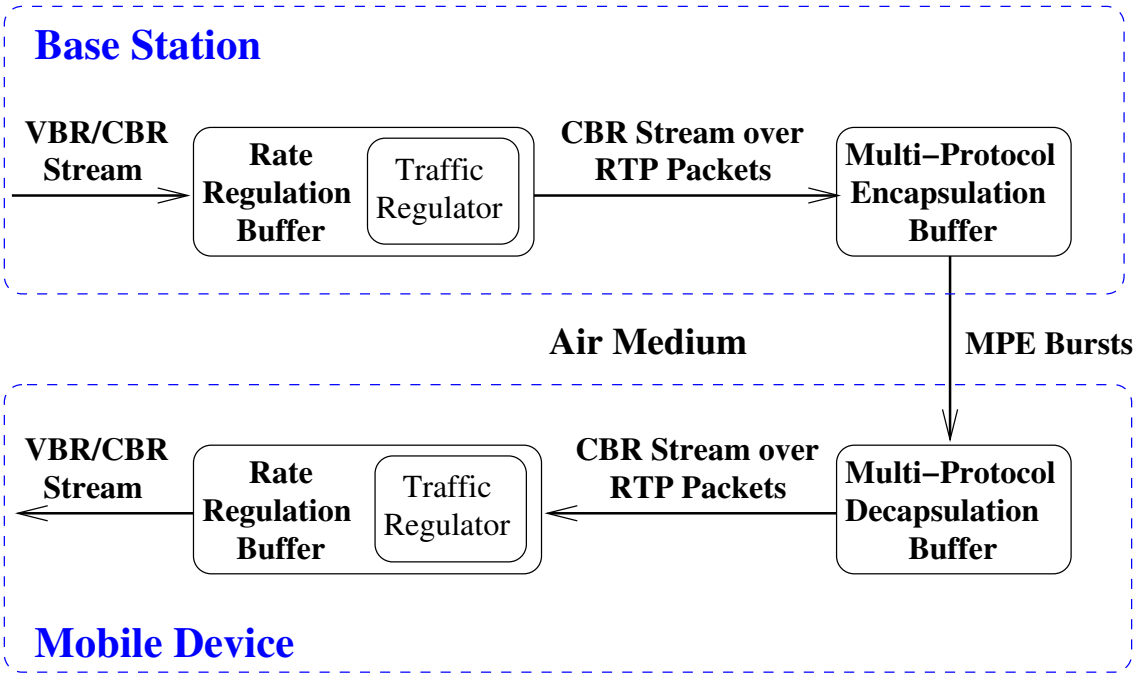


Figure 2.5: Rate regulator and other buffers in mobile video broadcast systems.

mobile device. Finally, the CBR stream is converted back to the original video stream by the traffic regulator in the Rate Regulation Buffer on the mobile device. The burst scheduling algorithm proposed in this thesis manages the Multi-Protocol Encapsulation Buffer, which resides *after* the Traffic Regulator at the base station. We mention that buffer layout similar to Figure 2.5 is mentioned in the DVB-H standard documents [63, Section 5.1.3], and are being used in real systems.

Finally, we discuss the issues of broadcasting rate regulated VBR streams. To get rate regulated VBR streams, we need to specify the streaming rates, buffer capacity, and initial delays at encoding time, which instruct video coders to constrain the size of each coded frame in order to avoid over/underflowing the rate regulation buffer. Broadcasting the resulting rate regulated VBR streams may lead to a couple of drawbacks. First, they are not *unconstrained* VBR streams, as the coded frame size may be limited by the leaky bucket tubes illustrated in Figure 2.4. Therefore, the resulting coding efficiency is lower than that of UVBR streams. Second, the rate regulation process requires a rate regulation buffer and incurs longer initial delay. The memory requirements and the initial delays are *user-specified*, and thus can be controlled in reasonable ranges. Therefore, using mobile video

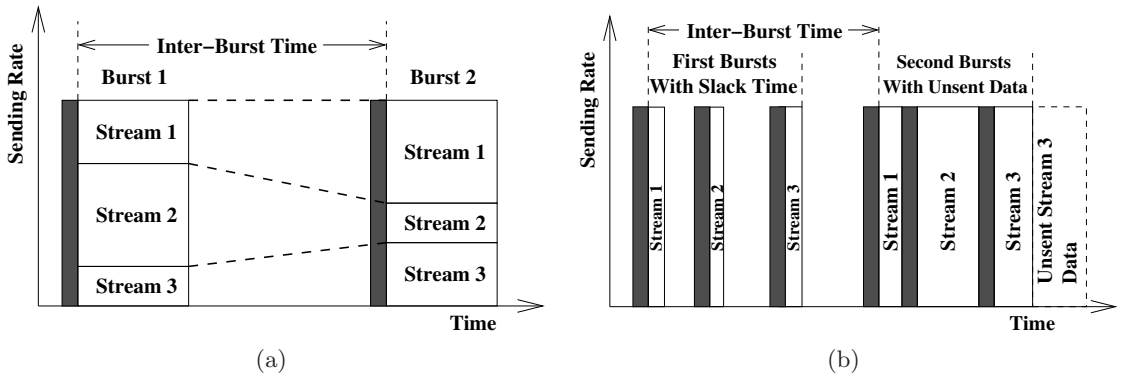


Figure 2.6: Video streams can be transmitted as (a) parallel and (b) serial services. Shaded areas represent overhead durations.

broadcast networks illustrated in Figure 2.5 to broadcast videos coded as rate regulated VBR streams is feasible and leads to better quality-of-service than broadcasting CBR streams. Nevertheless, a better solution is to directly broadcast unconstrained VBR streams for higher coding efficiency, shorter initial delays, and smaller memory requirements.

2.5 Burst Scheduling in Existing Base Stations

As mentioned in Section 2.3, for VBR video streams, multiplexers must ensure that the total bit rate of all video streams never exceeds the network bandwidth. Mobile video broadcast systems have already been deployed in Europe, Africa, and Asia, and in pilot-testing in several cities in North and South America [1]. The base stations in these deployments use commercial multiplexers such as UDCast IPE-10 [64], UBS DVE-6000 [65], and Grass Valley Opal II [66], which are some of the most popular multiplexers for DVB-H and DVB-SH (satellite services to handhelds) broadcast networks [8–10]. Most of the multiplexers implement two burst scheduling schemes: slotted scheduling that allocates fixed size bursts for each video stream, and dynamic scheduling that allows certain burst size flexibility, and requires a joint video coder to work. Slotted scheduling is the default burst scheduling algorithm because it is simple to implement, while dynamic scheduling is more complex and may not be available in low-end multiplexers. In this section, we study how to broadcast multiple VBR streams using these two scheduling schemes. Before that, we first review a simple way to broadcast VBR streams using parallel services.

2.5.1 Parallel Services

The multiplexers may support broadcasting multiple VBR video streams by considering each video stream as a parallel service. This is illustrated in Figure 2.6(a), in which the multiplexer combines three videos into two bursts. The multiplexer dynamically allocates different shares of bursts to individual video streams, based on their current bit rates. For example, as illustrated in in Figure 2.6(a), the multiplexer allocates more bits to stream 2 in burst 1, because stream 2 has a higher bit rate during burst 1. Similarly, the multiplexer assigns more bits to stream 1 in burst 2, because stream 1 has a higher bit rate during burst 2. While parallel services enable network operators to broadcast VBR streams, they incur high energy consumption on mobile devices, and thus shorten the battery life. This is because mobile devices have to turn on their receiving circuits for longer periods to receive all video streams sent in the same burst, despite that each mobile device only renders one of them. In contrast, sending every video stream as a serial service, as shown in Figure 2.6(b), enables each mobile device to only receive the video stream it is tuned to, and turn off its receiving circuit earlier. Since the parallel service leads to high energy consumption, we do not consider it in the rest of this thesis.

Figure 2.6(b) also illustrates that multiplexers must carefully construct burst schedules for higher energy saving and better bandwidth efficiency. More precisely, we make two observations on this figure. First, the first bursts of all three video streams are very short. Therefore, the overhead durations T_o become relatively large to these video streams, which result in high energy consumption and short watch time on mobile devices. Second, video stream 3 has too much data to send during its second burst, and some of its data, indicated by the dashed rectangle in this figure, cannot be transmitted although there is slack time among the first three bursts. Unsent data leads to late packets and playout glitches. These two observations show the importance of the burst scheduling problem in broadcast systems: an ill-formated burst schedule may lead to high energy consumption and/or low video quality.

2.5.2 Slotted Scheduling

Slotted scheduling algorithm schedules bursts in round-robin fashion, and can be used in open-loop broadcast systems. In slotted scheduling, 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

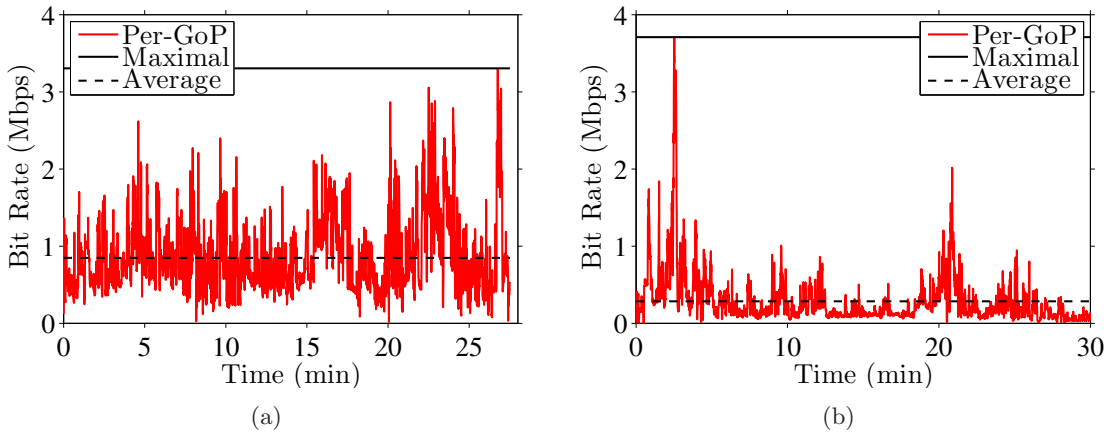


Figure 2.7: Per-GoP rate variations for two representative coded streams: (a) Silence of the Lambs, and (b) Tokyo Olympics.

multiplexer then schedules a burst every ΔT sec for every stream s, where each burst is b_s kb long. The slotted scheduling is very simple to implement, but it is not flexible because each video stream can only be transmitted in its allocated, fixed size bursts. More specifically, video streams that do not have enough accumulated video data to fill up its current burst would finish the transmission early, but the allocated time slot is not usable for other video streams. This leads to wasting of bandwidth. In addition, the multiplexers must drop some video data from those video streams that have more data to send than the sizes of their current bursts. This results in degraded streaming quality.

The slotted scheduling requires network operators to manually choose ΔT and b_s values to form a burst schedule that results in smooth playouts. The selections are often made through heuristics and are vulnerable to human errors. Selecting broadcast parameters for modern VBR coders is even more difficult, because these VBR coders trade high bit rate variations for better coding efficiency [49]. To illustrate the high rate variations, we choose two H.264/AVC coded video streams from the ASU Video Trace Library [67], and we plot the average bit rate of each GoP (Group of Pictures) in Figure 2.7. This figure indeed shows that real VBR streams consist of extreme rate fluctuations. We report GoP-level rate variations to be conservative: frame-level rate variations are even more severe.

There are two approaches to transmit VBR streams over constant bit-rate channels provided by the slotted scheduling: (i) directly sending each VBR stream at a heuristically chosen rate r_s, and (ii) adding a rate regulator for each video stream, and transmitting the

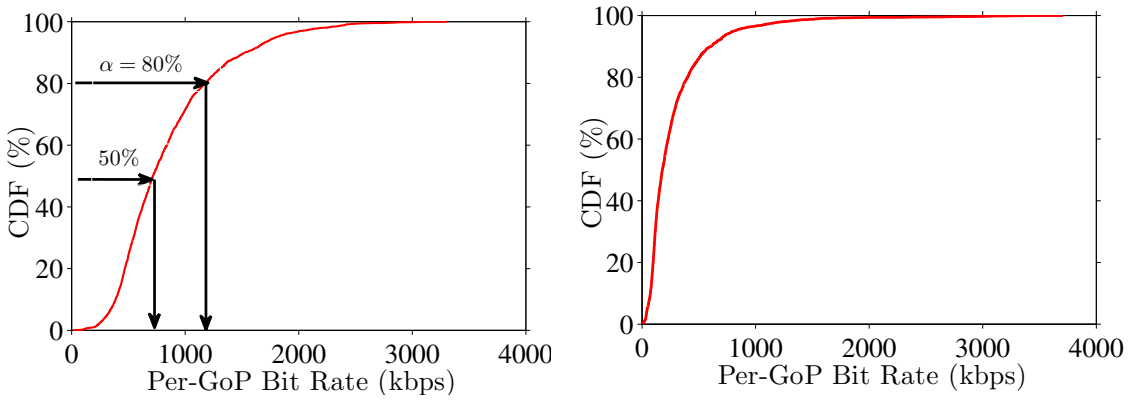


Figure 2.8: CDFs of Per-GoP bit rate of two representative coded streams: (a) Silence of the Lambs, and (b) Tokyo Olympics.

rate regulated VBR streams. We describe these two approaches in the following.

VBR (variable-bit-rate). We use r_s to denote the streaming rate of video stream s. Without loss of generality, we let $r_1 \leq r_2 \leq \cdots \leq r_S$; otherwise, we relabel the video streams. To ensure there is absolutely no buffer underflow instances, network operators have to set r_s to be the maximal bit rate of stream s, which is indicated by the solid line in Figure 2.7. Doing so, however, is too conservative and leads to oversubscription and wasting of bandwidth. Network operators may avoid the oversubscription issue by choosing a smaller r_s at the mean bit rate of that video stream, as indicated by the dashed line in Figure 2.7. This, however, may result in playout glitches due to buffer underflow instances on mobile devices. Therefore there exists a tradeoff between wasted bandwidth and video quality. To better quantify this tradeoff, we compute and plot the CDF (cumulative distribution function) curve $F_s(r)$ of per-GoP bit rate for the two considered streams in Figure 2.8. We then define a VBR burst scheduling algorithm VBR_α as streaming each video stream s ($1 \leq s \leq S$) at the smallest bit rate r_s so that $F_s(r_s) \geq \alpha$. For example, $\text{VBR}_{100\%}$ conservatively sets the streaming rate at the maximal bit rate, and $\text{VBR}_{50\%}$ sets the streaming rate at the mean bit rate.

Once the r_s is determined, we compute ΔT and b_s. We first choose a ΔT value as follows. We assume that mobile devices have a receiving buffer of Q kb. The ΔT value is bounded by $\Delta T \leq Q/r_S$, because longer ΔT would overflow the Q kb buffer of mobile devices that render video stream S. To achieve high energy saving, we set $\Delta T = Q/r_S$, as shorter ΔT

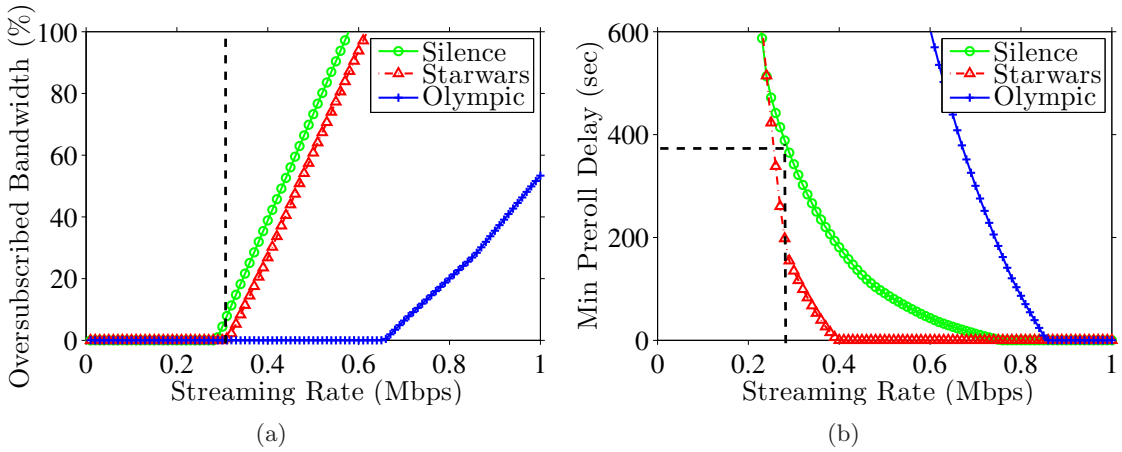


Figure 2.9: Regulating VBR streams into CBR streams using buffers may cause: (a) high oversubscribed, wasted, bandwidth, and (b) prohibitively long preroll buffering delay.

value leads to shorter bursts, and thus lower energy saving. We then divide the air medium time ΔT among all video streams proportionally to their bit rates. Mathematically, we write $b_s = R\frac{r_s}{\sum_{i=1}^{S} r_i}\Delta T$, where R kbps is the broadcast network bandwidth.

RVBR (regulated-variable-bit-rate). Traffic regulators absorb VBR traffic burstiness at the expense of higher memory requirements and longer preroll delays. *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 in the future. The preroll delay, unfortunately, could be prohibitively long and annoying to users. To illustrate, we analyze three video streams, and we use the H.264/AVC HRD model [62] to compute the buffer size and preroll delay requirements for any given streaming rate r_s. We notice that the leaky bucket model allows buffer underflow instances at the sender side [50], which is acceptable in Internet streaming systems because the Internet can use any oversubscribed bandwidth toward packets of other, background, traffics. However, oversubscribed bandwidth in broadcast networks is wasted, because the base station computes the burst schedule and reserves air medium time for individual video streams. To quantify this negative impact, we compute the oversubscribed bandwidth and minimum preroll delay at various r_s, and we plot the results in Figure 2.9. This figure indicates that long preroll delay is required if network operators want to avoid wasted bandwidth. 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 min* preroll delay, which is clearly not acceptable to users. Therefore, there exists a tradeoff

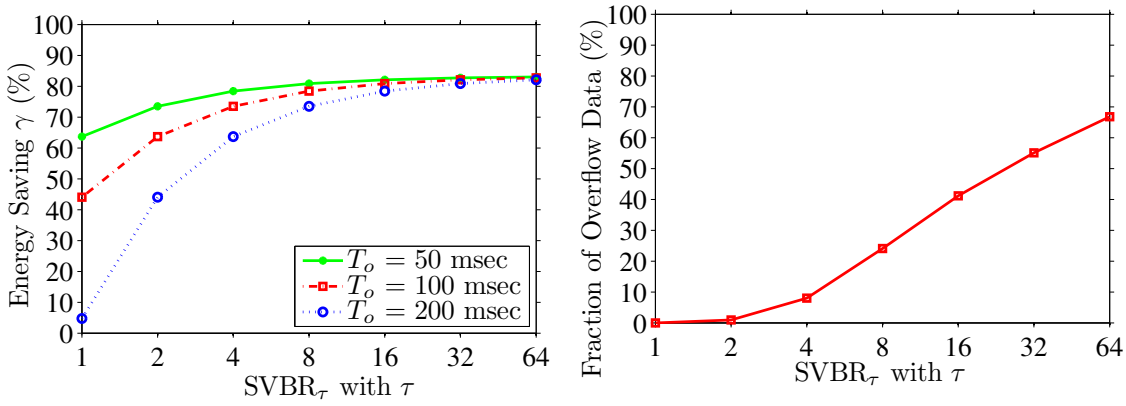
between wasted bandwidth and user experience. To better quantify this tradeoff, we define a burst scheduling algorithm RVBR_β as streaming each video stream s ($1 \leq s \leq S$) using a rate regulator. The regulated VBR stream has bit rate r_s that is the smallest bit rate such that $d_s(r_s) \leq \beta$ sec, where $d_s(r_s)$ represents the minimum preroll delay under streaming rate r_s. For each video stream s, $d_s(r_s)$ can be computed using the algorithm given in [62] as we show in Figure 2.9. For example, RVBR_3 sets r_s at the minimal streaming rate so that the preroll delay incurred by the rate regulation is less than 3 sec. Once the r_s is determined, ΔT and b_s can be computed as mentioned above.

2.5.3 Dynamic Scheduling

Multiplexers may support closed-loop broadcast networks using dynamic scheduling. Dynamic scheduling also allocates a burst for each video stream in round-robin fashion within each ΔT scheduling window. Unlike slotted scheduling, dynamic scheduling allows flexible burst size, i.e., video streams currently have higher bit rates may extend their bursts beyond what were reserved for them. However, bursts cannot span over more than one scheduling window. Within each scheduling window, the dynamic scheduling allocates each video stream a burst with size of its aggregate frame size in that scheduling window. Since closed-loop networks employ joint video coders to control the aggregate bit rate of all video streams, the air medium time of each scheduling window ΔT can accommodate all frames within that scheduling window without overloading the broadcast network.

Joint video coders, however, do not monitor the buffer states of mobile devices, and thus cannot ensure the mobile devices are free from buffer overflow instances. It is, therefore, network operators' responsibility to manually choose a proper ΔT value to avoid buffer overflow instances. We describe a very general approach to determine ΔT in the following.

DVBR (dynamic-variable-bit-rate). In general, larger scheduling window leads to longer bursts and thus higher chance to overflow receiving buffer on mobile devices. One way to avoid buffer 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. To cover a wide range of base station configurations, we define DVBR_τ as the dynamic scheduling algorithm with a scheduling window size $\Delta T = \tau Q/R$, where $\tau \geq 1$ is a system parameter. To quantify the implication of τ on system performance, we simulate a closed-loop broadcast network



(a) (b)

Figure 2.10: With different τ value, dynamic scheduling results in different: (a) energy saving and (b) fraction of overload.

with six H.264/AVC videos from the ASU Video Trace Library [67]. We notice that this Video Trace Library only provides trace files coded by standalone video coders. To mimic a typical joint video coder, we write scripts to allocate network bandwidth among these six videos using the Lagrangian optimization method [68], which allow us to maximize the average video quality under the constraint of limited network bandwidth. More specifically, for each video stream, we first use the trace files with different quantization parameters to derive its empirical R-D (rate-distortion) curves of each scheduling window. The empirical R-D curves allow us to predict the video quality at various encoding bit rates without going through time-consuming encoding process. Our scripts then compute the optimal bit rate allocation of all video streams in each scheduling window, and we use the resulting bit allocation as the video traces of a typical joint video coder.

We consider a broadcast network with 16.09 Mbps bandwidth, which is a common modulator configuration of DVB-H broadcast networks [16]. We assume mobile devices have 2 Mb receiver buffer. We concurrently broadcast six videos over this broadcast network for 10 minutes. We consider two performance metrics: energy saving, which is the fraction of time mobile devices can turn off their receiving circuits, and fraction of overflow data, which is the fraction of bursts that cannot fit into the receiver buffer. For each metric, we first compute the average performance of each video stream throughout the broadcast, we then report the mean value among all six video streams. We repeat the simulation multiple times, each with a different τ value from 1 to 64.

We plot the results in Figure 2.10. Figure 2.10(a) shows that the energy saving achieved under three typical overhead duration T_o: 50, 100, and 200 msec. This figure illustrates that mobile devices may suffer from low energy saving when τ is small. For example, when $\tau = 1$ and $T_o = 100$ msec, the energy saving is about 40%, which is only half of the 80% energy saving when $\tau \geq 16$. We plot the fraction of lost data in Figure 2.10(b). This figure reveals that larger τ value leads to more lost data, which results in playout glitches. For example, when $\tau = 16$, mobile devices on average lose 40% of the video data, which clearly would prevent the video streams from proper rendering. Figure 2.10 shows that, in dynamic scheduling, network operators must carefully choose the ΔT. Otherwise, the broadcast network may suffer from low energy saving and/or degraded user experience. More importantly, closed-loop networks suffer from buffer overflow instances unless τ is set to 1. Therefore, we assume that network operators adopt $\tau = 1$ for glitch-free broadcast, if not otherwise specified.

2.5.4 Summary

We described slotted scheduling and dynamic scheduling algorithms implemented in current commercial multiplexers. They both require network operators to manually choose broadcast parameters, which is time-consuming and error-prone. We defined three scheduling algorithms VBR_α, RVBR_β, and DVBR_τ, which simplify the process for network operators to choose broadcast parameters for current base stations. We also use these algorithms as the benchmarks of our proposed burst scheduling algorithms in Chapter 4.

2.6 Inferring Overhead Duration of Real Mobile Devices

With current technology, T_o is reported to be in the range of 50—250 msec [9, 10, 16], and for a specific example, the Philips mobile TV chip has an T_o of 150 msec [69]. In this section, we design and conduct an experiment to infer the *actual* overhead duration T_o of a real mobile device. We use this value in most of the experiments and analyses conducted in this thesis. We use Nokia N96 cellular phones in this experiment because it is the most recent Nokia phone that supports DVB-H at the time of writing. Hence, its T_o value can serve as a *lower bound* of other legacy, older mobile devices. In order to infer the T_o value of Nokia N96, we need to know the power consumption of its DVB-H chip before applying the time slicing technique, which is denoted by c mW. Unfortunately, the make and model of

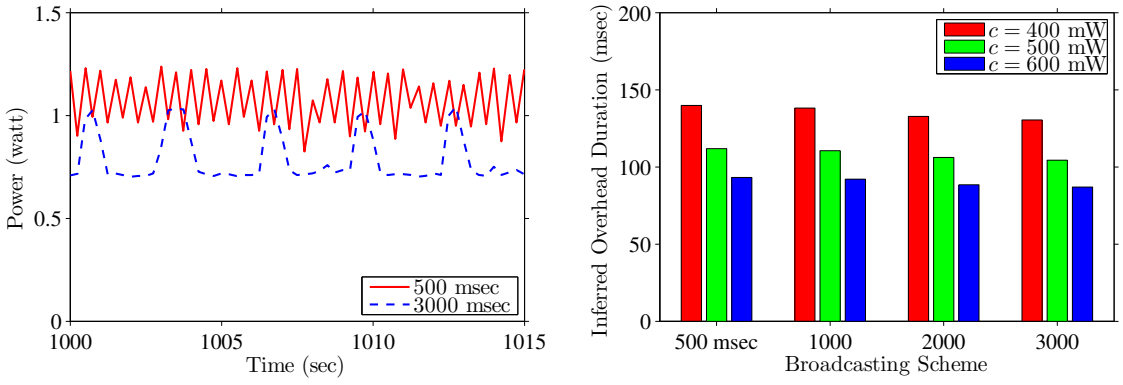


Figure 2.11: (a) Sample power consumption results captured on a Nokia N96 cellular phone, and (b) overhead duration inferred from different broadcast schemes and chip specifications.

the DVB-H chip used by Nokia N96 are not public. Therefore, we consider the data sheet released by a popular DVB-H chip manufacturer [18], which indicates the first generation of DVB-H chips have a power consumption of 700 mW, while the most recent ones have a power consumption of 400 mW. We consider $c = 400$, 500, and 600 mW in the experiment.

We conduct the experiment using a real mobile TV testbed that implements a DVB-H network. Details of this testbed are given in Section 3.6. We capture a 10-minute news clip over a digital cable service, and we encode the video using an H.264/AVC encoder at bit rate 450 kbps, and the audio using an eAAC+ encoder at 32 kbps. We use our mobile TV base station to broadcast the coded video stream, and we restart the stream upon the end of it is reached. We broadcast the video stream with a fixed inter-burst time period of 250 msec for 3.5 hrs. We cannot conduct longer experiments, because the battery of N96 only lasts for 3.5 hrs in some test scenarios. We use a Nokia N96 to watch this TV channel, and we measure its energy consumption using a built-in battery discharge monitor, called Juice. Juice is a light-weight monitor that runs in the background, measures battery voltage, current, as well as power consumption, and saves the results into a log file. A recent study reports that Juice is fairly accurate [70], compared to external instruments. Upon getting the log file for this 3.5-hr experiment, we fully charge the cellular phone battery. We then broadcast the same video with a different inter-burst time period. We consider five different broadcasting schemes with inter-burst periods: 250, 500, 1000, 2000, and 3000 msec, and we collect a log file for each of them.

We plot two sample curves of N96 power consumption in Figure 2.11(a); curves for other broadcasting schemes are similar and not shown for brevity. This figure illustrates that broadcasting schemes with shorter inter-burst time periods result in shorter sleep time, while schemes with longer inter-burst time periods lead to longer sleep time between bursts. Moreover, these log files allow us to compute the average power consumption of each broadcast scheme over the 3.5-hr broadcast time. Next, we draw an important observation: the power consumption difference between any two of the broadcasting schemes can be completely attributed to the number of overhead durations T_o they impose. This is because we broadcast exactly the same video stream using each of these broadcast schemes, so that the energy consumed by receiving, decoding, rendering, and displaying the video are the same for all of them. That is, if we use the 250 msec broadcast scheme as the baseline, we can derive the T_o value for each broadcast scheme using the energy consumption difference between them. As an illustrative example, consider a 1-sec period of time, the energy consumption difference between the 250 and 500 msec broadcast schemes is due to the *two extra overhead durations* of the 250 msec scheme, compared to the 500 msec scheme. Therefore, given a DVB-H chip energy consumption c, we can infer the T_o based on these two schemes. Similarly, we can infer the T_o by comparing 200 msec scheme with 1000, 2000, and 3000 msec schemes. We report the inference results in Figure 2.11(b). This figure reveals that, even with modern cellular phones such as Nokia N96, the overhead duration T_o is about 80–140 msec, which is non-trivial compared to burst lengths, which usually are a few hundreds msec long. The experimental results confirm the overhead duration range of 50–250 msec reported in the literature [9, 10, 16].

Chapter 3

Energy Optimization

In this chapter, we formally describe and formulate the energy optimization problem in mobile video broadcast networks. We then show its hardness and propose an optimal algorithm to solve a practical simplification of the general optimization problem. We also propose a near-optimal algorithm for the general problem. These two algorithms are analytically analyzed and throughly evaluated using simulations and experiments.

3.1 Introduction

We study the energy optimization problem described in Problem 1 (stated in Section 1.2.1), in which a base station concurrently broadcasts multiple video streams to mobile devices over a common wireless medium to many mobile devices. In such networks, base stations employ time slicing to save energy and they must carefully compose burst transmission schedules, which specify the burst start times and sizes, in order to guarantee streaming quality and proper functioning of the system. Despite the importance of burst scheduling, current base stations employ simple heuristics to construct burst schedules. For example, the base station in Nokia Mobile Broadcast Solution (MBS) [71] can only take a system-wide scheduling window length, which is manually specified by the network operator. That is, all video streams, despite their encoding bit rates, have the same number of bursts in every scheduling window, while the burst size of each video stream may be different. Manually selecting a scheduling window length and burst sizes for individual video streams with different bit rates is challenging and error-prone, and the resulting burst schedules may not be optimal in terms of energy saving. This is because when video streams have different

bit rates, the number of bursts that each video stream requires as well as the size of each burst will be different in optimal schedules. Therefore, each video stream should not only have *diverse* but also *dynamic* inter-burst periods, which are not supported by current base stations such as Nokia MBS. Hence, a more efficient burst scheduling (in terms of energy saving) is required.

In this chapter, we formulate the energy optimization problem in mobile video broadcast networks. We show that this problem is NP-complete for video streams with arbitrary bit rates. We then solve this problem in two steps. First, we propose a practical simplification of the general problem, which allows video streams to be classified into multiple classes and each class has a different bit rate. The bit rate of class c, r_c, can take any value in the form of $r_c = 2^i \times r_1$, where $i \in \{0, 1, 2, 3, \dots\}$, and r_1 is the bit rate of the lowest class. r_1 can take any arbitrary bit rate. For example, the bit rates 800, 400, 200, and 100 kbps could make four different classes for encoding sports events, movies, low motion episodes, and talk shows, respectively. Using the above simplification, we develop an optimal (in terms of energy saving) burst scheduling algorithm, which is quite efficient: its time complexity is $O(S \log S)$, where S is the total number of video streams.

Next, we propose a near-optimal algorithm for the general burst scheduling problem that does not require any assumption on the stream bit rates. This algorithm allows content providers to choose the appropriate encoding bit rates for different types of video content without being constrained by power-of-two bit rate increments. This in turn provides better bandwidth efficiency of the expensive wireless spectrum and thus more offered video streams, as well as higher perceived video quality and thus wider adoption of the mobile video broadcast networks. The proposed algorithm achieves all of the above while making the energy consumption of mobile devices very close to the absolute possible minimum. We evaluate both algorithms using real implications and simulations. We design and build a real mobile TV testbed, and we implement the proposed algorithms in the testbed. The experimental results from the testbed demonstrate the practicality and effectiveness of our proposed algorithms. We also conduct simulation experiments to analyze the performance of our algorithms under wide ranges of parameters.

3.2 Related Work

Several works have addressed energy saving in mobile video broadcast networks. The authors of [19] and [16] estimate the effectiveness of the time slicing for given burst schedules. Both works indicate that time slicing enables mobile devices to turn off their receiving circuits for a significant fraction of the time. These two works do not construct burst schedules, they only compute the achieved energy saving for given *pre-determined* burst schedules. In contrast, we formulate and solve the burst scheduling problem for arbitrary channel bit rates. To the best of our knowledge, there exist no other burst scheduling algorithms that can accommodate video streams at *different* bit rates in the literature.

The authors of [72] propose an energy saving strategy for DVB-H networks by not receiving some MPE-FEC sections once the received sections can successfully reconstruct the data. Skipping a few MPE-FEC sections means that mobile devices can turn off their receiving circuits earlier, which leads to additional energy saving compared to receiving all MPE-FEC sections regardless whether they are necessary. The authors of [73] consider mobile devices with an auxiliary short range wireless interface and construct a cooperative network among several receivers over this short range wireless network. Mobile devices share received IP packets over this short range network, so that each mobile device only receives a small fraction of IP packets directly from the DVB-H network. This allows receivers to reduce the frequency of turning on their receiving circuits. Assuming sending/receiving IP packets through the short range network is more energy efficient than receiving DVB-H sections, this cooperative strategy can reduce energy consumption. The proposals in [72,73] are orthogonal and complementary to ours , as they reside in the mobile devices themselves and try to achieve additional energy saving on top of that achieved by time slicing. In contrast, our proposed algorithms are to be implemented in the base station broadcasting video streams to mobile devices.

Optimizing mobile video broadcast networks from aspects other than energy saving has also been studied in the literature, including the radio performance optimization in [74], and the frame refresh delay reduction in [33, 75–77]. The author of [74] studies DVB-H radio performance using simulations. The DVB-H system parameters are classified into three sets: physical layer, time slicing module, and MPE-FEC module, while interaction among parameters in different sets are outlined. The simulation results in [74] indicate that extending the time interleaving depth in MPE-FEC to at least 100 msec results in good

radio link performance, while increasing it beyond 300 msec yields no further improvements. Extending the time interleaving depth, however, increases burst durations and imposes negative impact on power consumption.

The frame refresh delay refers to the time period between receiving the first bit of a new video stream and reaching the next random access point, typically an intra-coded frame, of that video. Shorter frame refresh delays lead to shorter channel switching delays, and thus more responsive systems. To reduce frame refresh delays, the authors of [75] propose to periodically add redundant intra-coded frames into video streams coded by H.264/AVC [78]. By frequently adding low quality intra-coded redundant frames into a video stream, more random access points are created, which in turn reduces the refresh delay. Instead of sending low-quality intra-coded frames over dedicated channels, intra-coded frames can also be transmitted at the beginning of bursts to shorten frame refresh delays [33, 76, 77].

None of the aforementioned works presents burst scheduling algorithms. In fact, unlike our burst scheduling problem which is in the link layer, the radio performance optimization lies in the physical layer, and both frame fresh delay reduction and video quality optimization fall in the application layer. Therefore, all these works are complementary to our work on maximizing energy saving.

Finally, we note that receivers in *broadcast* networks usually have separate receiving circuit and antenna for processing broadcast signals, other than the circuits for receiving and making phone calls. Our work focuses only on optimizing the energy saving for broadcast receiving circuits. In addition, because of the one-way nature of broadcast networks, feedback channels from numerous receivers to the base station are not practical. Thus, many of the energy saving techniques designed for video streaming to the general wireless devices are not applicable to mobile TV networks. For example, the throttling technique proposed in [79], which enables a wireless receiver to indirectly control the sending pattern of an Internet streaming server, requires a feedback channel from the receiver to the server, which may not be possible in one-way broadcast networks.

3.3 Notations and Hardness

In this section, we define notations for the energy optimization problem described in Problem 1. We then show that it is NP-complete. We list all symbols used in the chapter in Table 3.1 for quick reference.

Table 3.1: List of symbols used in energy optimization.

Sym	Definition	Sym	Definition
T_o	overhead duration	n_s	no. bursts for s
S	number of video streams	f_s^k	time of burst k for s
R	burst bit rate	b_s^k	burst size
Q	receiver buffer size	c_s^k	buffer level
r_s	bit rate for stream s	u_s	init. buffer level
p	scheduling window length	γ	energy saving
$\mathbf{L}$	burst schedule	d	ch. switching delay
$\mathbf{w}_s$	set of subframes for s	w_s^k	subwindow k for s
x_s^k	start time of w_s^k	z_s^k	end time of w_s^k
y_s^k	total burst time of w_s^k	e_s^k	completion time of w_s^k

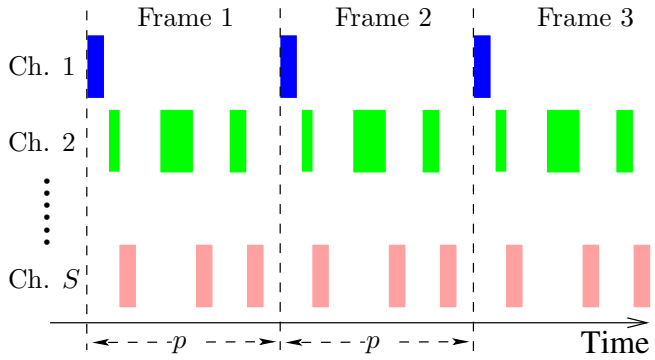


Figure 3.1: The burst scheduling problem in mobile video broadcast networks.

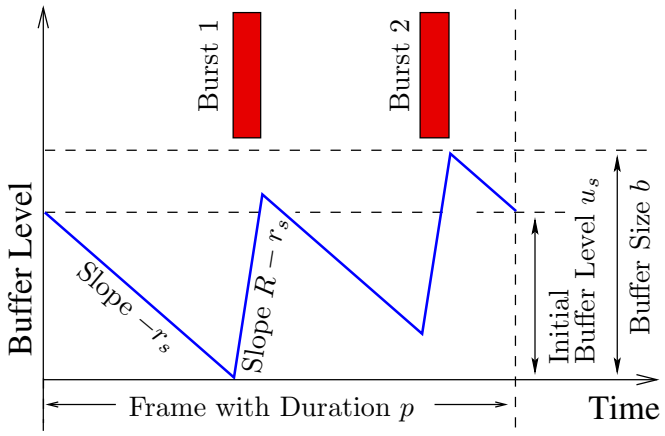


Figure 3.2: The dynamics of the receiver buffer during in successive frames.

We consider video broadcast networks in which a base station concurrently broadcasts S video streams to users with mobile devices over a wireless medium with bandwidth R kbps. Each video stream s, $1 \leq s \leq S$, has a bit rate r_s kbps, which is typically much less than R. The base station broadcasts each video stream in *bursts* at bit rate R kbps. After receiving a burst of data, the receiving circuits are switched off until the time of the next burst, which is computed by the base station and included in the header fields of the current burst. The receiving circuits of mobile devices must be open slightly before the burst time, because it takes some time to wake up and synchronize the circuitry before it can start receiving data. This time is called the overhead duration and is denoted by T_o. T_o is in the range of 50—250 msec [9,10,16], and in Section 2.6, we empirically show that a recent Nokia cellular phone has T_o in the range of 80–140 msec. The energy saving achieved by mobile devices receiving video stream s is denoted by γ_s and it is calculated as the ratio of time the video stream is in off mode to the total time [16, 19]. We define the system-wide energy saving metric over all video streams as $\gamma = \left(\sum_{s=1}^{S} \gamma_s\right)/S$. The energy saving as well as the burst scheduling itself are performed on a recurring time window called a *scheduling window*. We let p denote the scheduling window length, which is a system parameter in mobile video broadcast networks. Figure 3.1 shows an example of three recurring windows. In general, longer scheduling windows provide more chances to shuffle bursts around for better energy saving. However, a longer p may increase the channel switching delay and computation complexity of the burst scheduling algorithm. We will empirically study these tradeoffs in Section 3.9.

Figure 3.1 illustrates a simple example for the burst scheduling problem. Notice that the bursts have different sizes, are disjoint in time, and are repeated in successive frames. In addition, there can be multiple bursts for a video stream in each recurring scheduling window to ensure that there are no buffer under/overflow instances. To illustrate the receiver buffer dynamics for a valid solution of the burst scheduling problem, we demonstrate in Figure 3.2 the buffer level as a function of time. This is shown for a receiver of an arbitrary video stream s with two bursts in each scheduling window. We make two observations on this figure. First, during a burst, the buffer level increases with a rate (slope of the line) of $R-r_s$, which is much larger than the consumption rate of $-r_s$ when there is no burst. Second, the scheduling window starts with an initial buffer level (denoted by u_s) and ends at the same buffer level. Clearly, this is a requirement for any valid burst scheduling solution, otherwise there is no guarantee that the buffer will not suffer from over or underflow instances.

It is important to note that in video streaming systems, there are typically multiple buffers in different layers, and these buffers are coordinated using signaling mechanisms of the protocols in the corresponding layers. For example, in the application layer, an H.264/AVC video encoder can use the buffering model described in the standard document [53, Annex. C] to specify the buffer requirements for the decoder in order to have a smooth playout of the encoded video. The DVB-H standard, on the other hand, defines a buffering model for the physical, link, and application layers [80, Section 5.3]. As we mentioned in Section 3.2, our burst scheduling problem belongs to the time slicer in the link layer. Thus, the receiver buffer mentioned in Problem 1 and throughout the chapter is the link-layer buffer, which is also known as the time slicing buffer [81, Section 9.4].

We prove in the next theorem that the burst scheduling problem is NP-complete.

Theorem 1 (Burst Scheduling)**.** *The burst scheduling problem stated in Problem 1 is NP-complete.*

Proof. We first show that the problem of maximizing energy saving (Problem 1) is the same as the problem of minimizing the total number of bursts in each scheduling window for a given scheduling window length p. To maximize energy saving γ, we have to minimize the receiving circuit on-time for receivers. Notice that the receiving circuit on-time can be divided into two parts: burst and overhead durations as illustrated in Figure 1.1. The burst duration represents the time in which mobile devices receive the video data. Since we consider steady burst schedules, where the number of received bits is equal to the number

of consumed bits in each scheduling window for all mobile devices, the burst duration must be constant across all feasible burst schedules. Notice also that each burst incurs a fixed overhead T_o. Therefore, minimizing the receiving circuit on-time is equivalent to minimizing total number of bursts in each scheduling window, because it minimizes the total overhead.

Next, we reduce the NP-complete problem of task sequencing with release times and deadlines [82, pp. 236] to the problem of minimizing the total number of bursts in each scheduling window. The task sequencing problem consists of T tasks, where each task $t = 1, 2, \ldots, T$ is released at time x_t with length y_t sec and deadline z_t. The problem is to determine whether there is a single machine (non-preemptive) schedule that meets all constraints of release times and deadlines. For any task sequencing problem, we set up a burst scheduling problem as follows. We let $S = T$ and map every task to a video stream. We let $p = p^*$ be the optimum scheduling window length, which will be derived in the next section. We choose an arbitrary burst bit rate R. For any video stream s ($s = 1, 2, \ldots, S$), we let video stream bit rate $r_s = Ry_s/p^*$ to balance the number of received bits and the number of consumed bits. We set the initial buffer level $u_s = (z_s - y_s)r_s$, which guarantees that a burst with length y_s will be scheduled (and finished) *before* the deadline z_s, or mobile devices will run out of data for playout (underflow). We let the receiver buffer size $b_s = u_s + y_sR - (x_s + y_s)r_s - \epsilon$, where $\epsilon > 0$ is an arbitrary small number. Selecting such an b_s guarantees that a burst with length y_s will be scheduled *after* the release time x_s, otherwise mobile devices will run out of buffer space (overflow).

Clearly, we can set up the burst scheduling problem in polynomial time. Furthermore, solving the burst scheduling problem leads to the solution of the task sequencing problem because the minimum total number of bursts is equal to S if and only if there is a non-preemptive schedule that satisfies the constraints on release times and deadlines of the task sequencing problem. Thus, the burst scheduling problem is NP-hard. Finally, determining whether a given burst schedule meets the collision free and buffer violation free requirements, i.e., a valid solution for Problem 1, takes polynomial time. Hence, the burst scheduling problem is NP-complete. □

We notice that this theorem might seem counter intuitive at first glance, because the burst scheduling problem looks somewhat similar to preemptive machine scheduling problems. However, there is a fundamental difference between our burst scheduling problem and various machine scheduling problems: most of the machine scheduling problems consider

costless preemption model [83]. In contrast, our burst scheduling problem adopts *costly* preemption model, as our problem aims at minimizing the total number of bursts in a scheduling window, which is essentially the number of preemptions. Therefore, the algorithms developed for various machine scheduling problems are not applicable to our burst scheduling problems (see [83] for a comprehensive list of machine scheduling problems). The costly preemption model has only been considered in a few works [84–87]. The authors of [85, 86] partially cope with preemption costs by adding constraints to limit the number of preemptions. The authors of [84] solve the problem of minimizing the weighted sum of the total task flow time and the preemption penalty, where the weight is heuristically chosen. The author of [87] considers the problem of minimizing weighted completion time and task makespan under a given preemption cost. Unlike these problems, our burst scheduling problem solely uses the preemption cost as the objective function and does not allow any late task, which renders the algorithms proposed in [84–87] inapplicable to our problem.

3.4 Problem Formulation

We show a simple example of the burst scheduling problem in Figure 3.1. This figure depicts that, in the considered problem, the bursts have various sizes, are disjoint in time, and are repeated in all recurring frames. Furthermore, a video stream can have multiple bursts in each scheduling window to ensure that there is no buffer violations. To illustrate the receiver buffer dynamics for a valid schedule, we demonstrate in Figure 3.2 the receiver buffer level of a video stream with two bursts in each scheduling window. We make two observations on this figure. First, during a burst, the buffer level increases with a rate (slope of the line) of $R - r_s$, which is much larger than the consumption rate of $-r_s$ when there is no burst. Second, the scheduling window starts with an initial buffer level (denoted by u_s) and ends at the same buffer level. Clearly, this is a requirement for any valid burst scheduling solution, otherwise the receiver buffer may have over/underflow instances.

Let n_s be the number of bursts of video stream s in each scheduling window. We denote the start time and burst size of burst k of stream s as f_s^k sec and b_s^k kb, respectively, where $s = 1, 2, \ldots, S$ and $k = 1, 2, \ldots, n_s$. Since mobile devices open their receiving circuits T_o msec before f_s^k and it takes b_s^k/R to transfer b_s^k kb data, the receiving circuits are on for burst k of stream s during time period $[f_s^k - T_o,\ f_s^k + b_s^k/R)$. In addition, any burst b_s^k must be smaller than the receiver buffer size Q, i.e., $0 < b_s^k \leq Q$. We define the buffer level at

the beginning of burst k of video stream s as c_s^k kb. As illustrated in Figure 3.2, c_s^k can be computed as:

$$c_s^k = u_s + \sum_{i=1}^{k-1} b_s^i - f_s^k r_s,$$

where the second term accounts for the received data and the third term accounts for the consumed data. The output of the burst scheduling algorithm is a schedule $\mathbf{L} = \{\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_S\}$, where $\mathbf{L}_s$ is the schedule for video stream s. In addition, $\mathbf{L}_s = <n_s, u_s, \mathbf{f}_s, \mathbf{b}_s>$, where n_s is the number of bursts, u_s is the initial buffer level, $\mathbf{f}_s = \{f_s^1, f_s^2, \ldots, f_s^{n_s}\}$ indicates the burst start times, and $\mathbf{b}_s = \{b_s^1, b_s^2, \ldots, b_s^{n_s}\}$ represents the burst sizes.

The burst scheduling problem can then be formulated as:

$$\max_{\mathbf{L}} \quad \gamma = \sum_{s=1}^{S} \left(1 - \sum_{k=1}^{n_s} (T_o + b_s^k/R)/p\right) \Big/ S \tag{3.1a}$$

$$\text{s.t.} \quad \left[f_s^k, f_s^k + \tfrac{b_s^k}{R}\right) \bigcap \left[f_{\bar{s}}^{\bar{k}}, f_{\bar{s}}^{\bar{k}} + \tfrac{b_{\bar{s}}^{\bar{k}}}{R}\right) = \varnothing; \tag{3.1b}$$

$$c_s^k \geq 0; \tag{3.1c}$$

$$c_s^k + b_s^k - \tfrac{b_s^k}{R} r_s \leq Q; \tag{3.1d}$$

$$0 \leq u_s \leq Q; \tag{3.1e}$$

$$\textstyle\sum_{i=1}^{n_s} b_s^i = p r_s; \tag{3.1f}$$

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

The goal is to compute the schedule $\mathbf{L}$ to maximize the objective function in (3.1a), i.e., the system-wide energy saving γ. The constraints (3.1b)–(3.1f) guarantee that the resulting burst schedule is feasible as defined in Problem 1. In particular, (3.1b) ensures that there are no burst intersections among all S streams. (3.1c) validates the buffer level for stream s at the start time of every burst to prevent buffer underflow instances. We note that c_s^k is a function of f_s^k, b_s^k, and u_s as defined above. (3.1d) validates the buffer level for stream s at the end time of every burst to prevent buffer overflow instances. It is sufficient to check the buffer level only at the start and end times, because the buffer level only increases during the bursts as illustrated in Figure 3.2. The buffer under/overflow instances at scheduling window boundaries are prevented by (3.1e). (3.1f) says that the number of received and consumed bits for stream s are equivalent in every scheduling window, which in turn ensures that the buffer level at the end of every scheduling window is equal to the initial buffer level u_s.

3.5 Optimal Solution for a Practical Simplification

In Section 3.3, we proved that the general burst scheduling problem is NP-complete. In this section, we present an algorithm that optimally and efficiently solves this problem under a certain assumption that usually holds in practice. To simplify the presentation, we first describe an overview of the algorithm and an illustrative example. We then analyze the algorithm and prove its correctness, optimality, and efficiency. Then, we analyze the tradeoff between the achieved energy saving and the channel switching delay. Finally, we discuss several practical issues of the proposed algorithm.

3.5.1 Overview

We propose an optimal algorithm for the burst scheduling problem when the bit rate of a video stream s, $1 \leq s \leq S$, is given by $r_s = 2^i \times r_1$ for any i, where $i \in \{0, 1, 2, 3, \dots\}$, and r_1 can be any arbitrary bit rate. As mentioned in Section 3.1, the video streams can be divided into classes, where each class contains similar-type multimedia content encoded at the same bit rate. Without loss of generality, we assume that the bit rates of the S streams are ordered such that $r_1 \leq r_2 \leq \cdots \leq r_S$. If otherwise, a re-labeling based on the bit rates is applied. We also assume that the bandwidth of the wireless medium satisfies $R = 2^k \times r_1$, where k is a positive integer. We present in Figure 3.3 an optimal algorithm for solving the burst scheduling problem in this case.

The basic idea of our algorithm is as follows. The algorithm first computes the optimal value for the scheduling window length p^* (We derive p^* in Theorem 3). It then divides p^* into bursts of equal size $p^* r_1$ bits. Thus, there are $(p^* R)/(p^* r_1) = R/r_1$ bursts in each scheduling window. Then, each video stream is allocated a number of bursts proportional to its bit rate. That is, video stream s, $1 < s \leq S$, is allocated r_s/r_1 bursts and video stream 1 is allocated only one burst in each scheduling window. Moreover, bursts of video stream s are equally spaced within the scheduling window, with inter-burst distance of $p^*/(r_s/r_1)$ sec. This ensures that there will be no underflow instances in the receiver buffer, because the consumption rate of the data in the buffer for video stream s is r_s bps and the burst size is $p^* r_1$ bits. Since the optimal scheduling window length can be written as $p^* = b/r_1$, the size of each burst is $p^* r_1 = b$, which is no larger than the receiver buffer size b. This ensures that there is no buffer overflow instances. Finally, bursts of different video streams are arranged such that they do not intersect in time, that is, the resulting schedule is conflict

free.

To achieve the above steps in a systematic way, the pseudo code in Figure 3.3 works as follows. It builds a binary tree bottom-up. Leaf nodes representing video streams are created first, where the leaf node of video stream s is annotated with the value r_s/r_1. The algorithm uses this value as the key, and inserts all leaf nodes into a priority queue. This priority queue is implemented as a binary heap to efficiently find the node with the smallest key. The algorithm then repeatedly merges the two nodes that have the least key values into a new internal node. This new internal node has a key value equivalent to the sum of the key values of its children. This is done by popping the smallest two values from the heap, and then pushing the newly created node into it. The merging of nodes continues till the tree has a height of $\log(R/r_1)$. The last merged node becomes the root of the binary tree. Note that if the wireless medium is fully utilized by the video streams, i.e., $\sum_{s=1}^{S} r_s = R$, the computed bursts of the different video streams will completely fill the scheduling window p. If otherwise (i.e., $< 100\%$ utilization), the wireless medium will have to be idle during some periods within the scheduling window. The algorithm represents these idle periods as dummy nodes in the tree.

Once the binary tree is created, the algorithm constructs the burst schedule. It allocates to each video stream a number of bursts that is equal to its key value. In order to ensure conflict-free schedule, the algorithm computes the start time for each burst as follows. For each leaf node representing a video stream, the algorithm traverses the tree top-down. During the traversal, each node is assigned the reverse bit pattern from the root to this node, where the right branch has the bit 1 and the left branch has the bit 0. The bit pattern for a leaf node encodes the number of bursts and their start times for the video stream corresponding to that node. For example, in a tree of depth 3, the bit pattern 010 means that the video stream is assigned only the second burst in a scheduling window of eight bursts. For leaf nodes at levels less than the depth of the tree, the bit pattern is padded with one or more 'x' to the left. For example, if a leaf node has the bit pattern x01 in a tree of depth 3, this means that the node is at level 2 from the root. It also means that this node should be assigned the two bursts: 001 and 101. Notice that the first burst assigned to any video stream is equal to the numeric value of its bit pattern, with all 'x' bits set to zero. The algorithm computes this value and refers to it as the offset. The algorithm then computes successive bursts relative to this offset.

Power-of-Two Optimal (P2OPT) Algorithm

1. Compute optimal scheduling window length p^*
2. Allocate a leaf node l for every stream s, **let** $l.ch = s$
3. Push all leaf nodes to a priority queue P with key r_s/r_1
4. **while true**{
5. **let** m_1 = pop_min(P), m_2 = pop_min(P);
6. **if** m_2 is **null** or m_1.key < m_2.key { // no sibling
7. **if** m_2 is not **null** push(P, m_2); // return m_2 back to P
8. Allocate a dummy node m_2, where m_2.key = m_1.key
9. }
10. Create an internal node n with children n.left & n.right
11. **let** n.left = m_1, n.right = m_2;
12. **let** n.key = m_1.key + m_2.key;
13. push(P, n); // insert this internal node
14. **if** n.key $\geq R/r_1$ **break**;
15. }
16. **if** $|P| > 1$ **return** $\varnothing$; // no feasible schedule
17. **let** $\mathbb{T} = \varnothing$; // start composing schedule $\mathbb{T}$
18. Traverse the tree from root down, annotate each node with an *offset* with the value
18. of the reverse bit pattern from root till this node
19. **foreach** leaf node l {
20. **for** $i = 0$ to l.key - 1 {
21. start = offset + $i \times (R/r_1)/l$.key;
22. // add a burst to stream l.ch at time start $\times\ p^* r_1/R$
23. insertBurst($\mathbb{T}$, start $\times\ p^* r_1/R$, l.ch);
24. }
25. }
26. **return** $\mathbb{T}$;

Figure 3.3: An optimal algorithm to solve the burst scheduling problem.

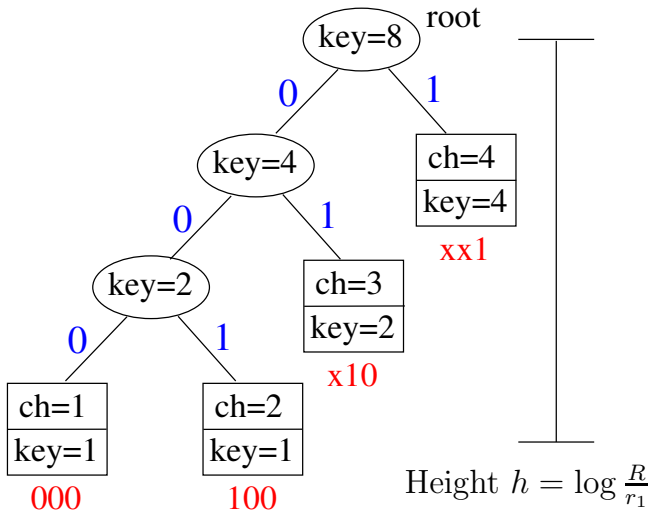


Figure 3.4: An illustrative example for the P2OPT algorithm.

3.5.2 Illustrative Example

Consider four video streams distributed over three different classes: two streams in class I with $r_1 = r_2 = 256$ kbps, one in class II with $r_3 = 512$ kbps, and one in class III with $r_4 = 1024$ kbps. Let the wireless medium bandwidth $R = 2048$ kbps, and the receiver buffer $b = 1$ Mb. As explained later, p^* is given by $p^* = b/r_1 = 4$ sec. The algorithm divides each scheduling window into $R/r_1 = 8$ bursts, and assigns 1, 1, 2, 4 bursts to streams 1, 2, 3, 4, respectively. The algorithm constructs a binary tree bottom-up as shown in Figure 3.4. Four leaf nodes are created, each representing a video stream and has a key value equal to the number of bursts that should be allocated to that stream. Notice that the leaf nodes are logically placed at different levels based on their key values. Then the algorithm recursively merges nodes with the same key values till it creates the root node with key value $8 = R/r_1$. Then, the algorithm constructs the schedule by traversing the tree from the root down to assign bit patterns to leaf nodes, which are shown in Figure 3.4. Using these bit patterns, the offset for each node is computed and the bursts are assigned. The resulting burst schedule is: $\mathbb{T} = \{(0.0, 1), (0.5, 4), (1.0, 3), (1.5, 4), (2.0, 2), (2.5, 4), (3.0, 3), (3.5, 4)\}$, where the first element in the parentheses is the start time of sending the burst, and the second element indicates the video stream.

Notice that we present a rather simple example for illustration, and the P2OPT algorithm is quite flexible on the bit rates of individual video streams. In fact, network operators

can broadcast each video stream s ($2 \leq s \leq S$) at an average bit rate $r_s = 2^i \times r_1$ for *any* i, where $i \in \{0, 1, 2, 3, \dots\}$. For instance, by setting $i = 0$ for all video streams, network operators can compute the optimal broadcast schedule for video streams with uniform bit rate. Therefore, the P2OPT algorithm provides a systematic way to construct optimal burst schedules for mobile broadcast networks that broadcast streams at uniform bit rates, which is currently a common practice.

3.5.3 Analysis

We show the correctness, efficiency, and optimality of our algorithm in the following two theorems.

Theorem 2 (Correctness and Efficiency)**.** *The burst scheduling algorithm P2OPT in Figure 3.3 returns a conflict-free schedule with no buffer under/overflow instances, if one exists. And it has a worst-case time complexity of $O(S \log S)$, where S is the number of video streams.*

Proof. We prove the correctness part in two steps. First, observe that a burst schedule produced by P2OPT is conflict-free because the algorithm assigns each video stream a unique bit pattern that specifies the allocated bursts to that video stream. Moreover, the bit pattern is padded with zero or more 'x' to the left, which guarantees that video stream s is assigned r_s/r_1 equally-spaced bursts. Hence, if P2OPT returns a schedule, this schedule is conflict-free with no buffer under/overflow instances.

Second, we prove that if P2OPT fails to return a schedule, there exists no feasible schedule for the given video streams. P2OPT only merges nodes at the same binary tree level and nodes at lower levels have strictly smaller key values than nodes at higher levels. Therefore, P2OPT merges all nodes from bottom-up and creates at most one dummy node at every level. Moreover, P2OPT uses line 12 to ensure the key value of each node indicates how many time slots are consumed by itself (for a leaf node) or by all leaf nodes in its subtree (for an internal node). P2OPT returns no feasible solution for a given problem if a full binary tree with height $h = \log(R/r_1)$ is built and $|P| > 1$ in line 16. Let z be the first merged node with key value R/r_1, which is the last merged node before returning from line 16. Since $|P| > 1$, we let $w \neq z$ be an arbitrary node in P. w must have key value no less than $\frac{1}{2}R/r_1$, otherwise w would have been merged before the children of z. We account for the number of time slots consumed by real (non-dummy) leaf nodes in subtrees beneath w

and z. w resides either at the same or higher level than z (case I), or at a level lower than z (case II). In case I, we know that $w.\text{key} \geq z.\text{key} = R/r_1$ and w is a real leaf node, because z is the first merged node at its level. Since P2OPT guarantees that at most one dummy leaf node exists at each level, the total time slots occupied by dummy leaf nodes in z's subtree cannot exceed

$$\sum_{i=0}^{\log R/r_1 - 1} 2^i = R/r_1 - 1$$

time slots. This shows that w and z consume at least

$$2\frac{R}{r_1} - (\frac{R}{r_1} - 1) = R/r_1 + 1$$

time slots. Since $R/r_1 + 1$ exceeds the total number of available time slots R/r_1, there exists no feasible schedule. Case II can be shown in a similar way and is omitted for brevity.

For time complexity, P2OPT can be efficiently implemented using a binary heap, which can be initialized in time $O(S)$. Notice that we have at most $\log R/r_1$ dummy leaf nodes because there is at most one dummy leaf node for each level. The while loop in lines 4–15 iterates at most $O(S + \log R/r_1) = O(S)$ times, because $\log R/r_1$ can be considered as a constant for practical encoding bit rates. Since each iteration takes $O(\log S)$ steps, the while loop takes $O(S \log S)$ steps. Constructing the burst schedule in lines 17–25 takes $O(S)$ steps, since the tree has up to $2S$ nodes. Thus, the time complexity of P2OPT is $O(S \log S)$. □

The following theorem shows that the P2OPT algorithm produces optimal burst schedules in terms of maximizing energy saving.

Theorem 3 (Optimality)**.** *The scheduling window length $p^* = b/r_1$, where b is the receiver buffer size, computed by P2OPT maximizes the average energy saving γ over all video streams.*

Proof. We leverage the fact that we established in the proof of Theorem 1: the problem of maximizing energy saving is equivalent to the problem of minimizing the total number of bursts in each scheduling window for a given scheduling window length p. To maximize energy saving γ, we have to minimize the ratio of receiving circuit on-time to the scheduling window length. We again divide receiving circuit on-time into burst and overhead durations. Notice that the ratio of burst duration to scheduling window length is constant for any feasible schedule because each video stream is broadcast at a given bit rate. Since each burst incurs overhead T_o sec, following the definition of γ, our burst scheduling problem

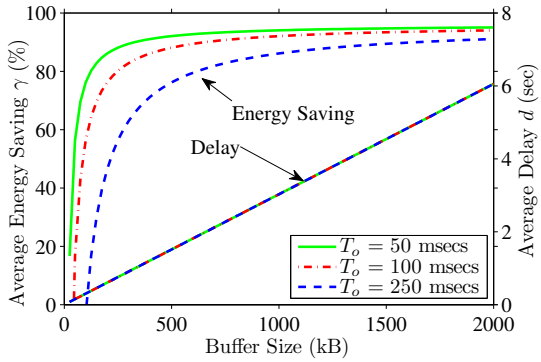


Figure 3.5: The trade-off between energy saving and channel switching delay.

is reduced to the problem of minimizing the total number of bursts normalized to the scheduling window length p.

We first prove by contradiction that any scheduling window length $p < p^*$ cannot result in higher energy saving. Assume a feasible schedule $\mathbb{T}_1$ results in higher energy saving with scheduling window length p_1 than schedule $\mathbb{T}^*$ with scheduling window length p^*, which is produced by P2OPT, where $p_1 < p^*$. To outperform $\mathbb{T}^*$, $\mathbb{T}_1$ must have fewer number of bursts than $\mathbb{T}^*$, i.e., $|\mathbb{T}_1| \leq |\mathbb{T}^*|$. However, $\mathbb{T}_1$ must lead to some buffer overflow, because fewer number of bursts is equivalent to longer bursts, but P2OPT has fully utilized (filled up) the receiver buffer b for all bursts. This contradicts the assumption that $\mathbb{T}_1$ is a feasible schedule.

We next consider $p > p^*$. Assume $\mathbb{T}_2$ is a feasible schedule that results in better energy saving with scheduling window length p_2, where $p_2 > p^*$. Let $k = \lceil p_2/p^* \rceil$. Define $\mathbb{T}'$ by repeating $\mathbb{T}^*$, which is produced by P2OPT, k times. Since burst schedule $\mathbb{T}'$ also fills up receiver buffer in all bursts, following the same argument as above, we show that $\mathbb{T}_2$ must lead to some buffer overflow. This contradicts the assumption that $\mathbb{T}_2$ is a feasible schedule. □

3.5.4 Tradeoff between Energy Saving and Switching Delay

We first compute the average energy saving γ^*. Since video stream s has an inter-burst duration of b/r_s and the burst length is b/R, the average energy saving is given by:

$$\gamma^* = \sum_{s=1}^{S} \Big(1 - r_s(1/R + T_o/b)\Big) \Big/ S = 1 - \sum_{s=1}^{S} r_s \frac{1/R + T_o/b}{S}. \tag{3.2}$$

Next, we analyze another important metric in mobile video broadcast networks: the channel switching delay d, which is the time a user waits before s/he starts viewing a selected channel when a change of channel is requested by her/him[1]. Channel switching delay is composed of several parts, in which the frame refresh delay and time slicing delay are the two dominating contributors [29,33]. In Section 3.2, we defined the frame fresh delay and described several works in the literature on minimizing it. The frame refresh delay is controlled in the application layer and is orthogonal to our burst scheduling problem. The time slicing delay refers to the time period between locking onto the radio signals and reaching the first burst of the subject video stream. Since time slicing delay is a *by-product* of the time slicing based energy saving scheme, we only consider time slicing delay in our work. We assume all other parts of channel switching delays are constant as they are outside of the scope of this chapter. The average channel switching delay due to the optimal schedule γ^* is then given by:

$$d^* = \Big(\sum_{s=1}^{S} b/2r_s\Big) \Big/ S = \Big(\sum_{s=1}^{S} 1/r_s\Big)(b/2S). \tag{3.3}$$

Notice that there is a tradeoff between γ and d and the main control parameter is the buffer size b. To examine this tradeoff, we present an illustrative example. We consider a broadcast service with $R = 10$ Mbps and 25 video streams equally distributed in five classes of heterogeneous bit rates (i.e., 1024, 512, 256, 128, and 64 kbps). We plot the energy saving and channel switching delay under different overhead durations in Figure 3.5. This figure shows that larger buffer sizes and smaller overhead durations lead to higher energy savings *and* also to higher channel switching delays.

[1]Each channel carries a video stream. We interchangeably use the terms "channel" and "stream" in this thesis.

3.5.5 Discussion

We discuss several practical issues related to the burst scheduling problem and our proposed P2OPT algorithm. First, we do not restrict the coding of videos to the exact power-of-two bit rates. We do not even restrict them to be CBR (constant-bit-rate) coded. What we assume is that network operators will broadcast the video over the closet power-of-two bandwidth, and smoothing buffers at the base station will be used to regulate the traffic. VBR streams achieve higher coding efficiency compared to CBR coded ones, and users of recent video coding standards, such as H.264/AVC, no longer use strict CBR coding [50]. Instead, users usually specify the network bandwidth and smoothing buffer parameters at encoding time, and the video coders perform rate control to ensure the resulting VBR coded streams can be transmitted over the CBR network channel without any buffer under/overflow instances. Therefore, our algorithm allows video streams to be VBR-coded and transmitted over CBR network channels using smoothing buffers, which is currently a common practice. Nevertheless, as mentioned in Section 2.4, smoothing buffers result in longer initial delay and higher memory requirements. Therefore, we directly solve the problem of broadcasting VBR streams in Chapter 4.

Second, we need to recompute the burst schedule many times during the broadcast in order to cope with the broadcast service dynamics. For example, each TV channel frequently changes programs (say each 30 min). Clearly different programs may have different video characteristics, and thus we need to reschedule whenever a change of program occurs on any TV channel. Given that there could be 20–50 TV channels concurrently broadcast over the same wireless medium, there will be a very high-level of dynamic changes that must be considered in real time by the burst scheduling algorithm. In addition, even during one TV program on a single TV channel, there are typically many commercial ads. Each commercial is a different video clip with different characteristics and bit rate. Furthermore, during a single commercial period, ads (video clips) change very fast, on the order of 30–60 sec. Each change *on any TV channel* triggers rescheduling. Hence, the efficiency of the proposed P2OPT scheduling algorithm is important. A brute-force approach to compute burst schedules for 20–50 channels may take prohibitively long time and may not even be doable at all, as the complexity is exponential in the total number of bursts for all channels.

Third, our P2OPT algorithm constructs transmission schedules that minimizes the energy consumption for mobile devices, and hence maximizes the viewing time for uses. Currently, there are some heuristic methods used in practice to construct burst schedules. For example, in Nokia's Mobile Broadcast Solution (MBS) [71, 88], network operators specify an inter-burst time period p sec, and a burst size b_s kb for each video stream s. The base station then schedules a burst every p 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 leads to no burst overlapping in time and no buffer overflow instances on mobile devices. This task is time consuming and error-prone. More importantly, the resulting energy saving is not maximized.

Finally, as mentioned in Section 2.3, video streams are typically obtained from multiple sources, e.g., from news, movies, and sports channels. It is also common that the operators of broadcast networks decode and then re-encode video streams to customize them for their networks and meet the limitations on the bandwidth of the allocated wireless spectrum. In addition, network operators have the option to broadcast video streams at different qualities. For example, sports channels can be broadcast at higher quality because they are watched by many users. Our proposed algorithm allows this differentiation in quality.

3.6 Design and Implementation of a Mobile TV Testbed

This section provides the details of the hardware and software components of a real mobile TV testbed[2]. We use this real testbed to evaluate the algorithms proposed in this and other chapter, in order to show their practicality and effectiveness. Figure 3.6 shows a picture of the testbed in our laboratory. We divide our description of the testbed into three parts: base station, mobile receivers and data analyzers, and signaling and electronic service guide. Each part is described in one of the following three subsections.

We note that our design is modular with well-defined interfaces between the components. Therefore, different hardware/software components can be updated with minimal effects on the others. Thus, we believe our testbed implementation will be useful for future generations of mobile TV systems.

[2]This testbed was designed and implemented in collaboration with Yi Liu and Cong Ly who are also graduate students of my advisor, Dr. Mohamed Hefeeda.

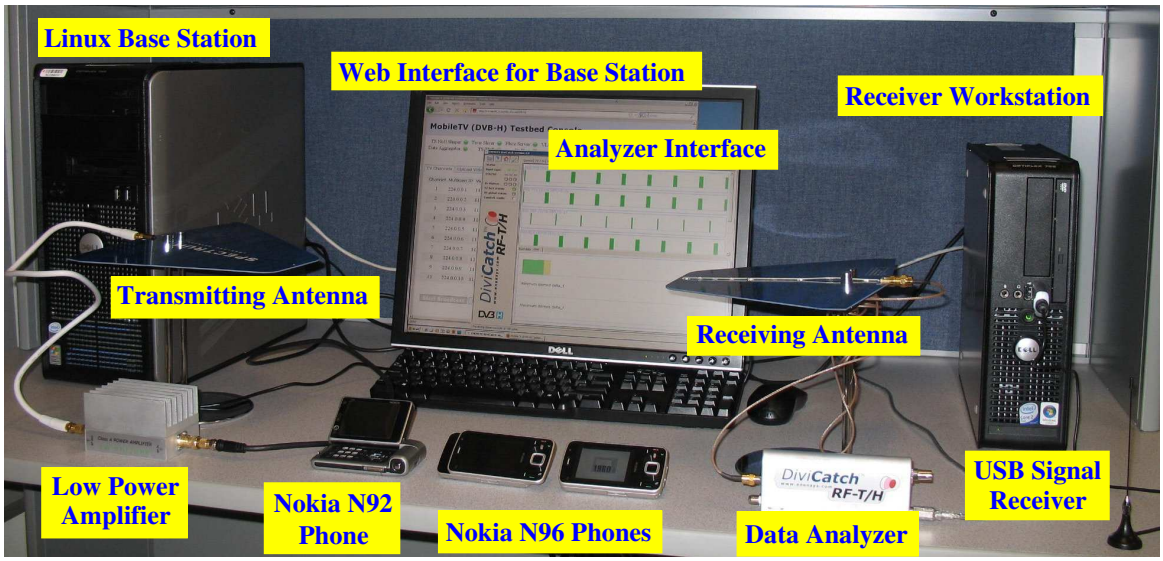


Figure 3.6: The hardware setup of the mobile TV testbed. Left: the base station; Right: the receivers/analyzers.

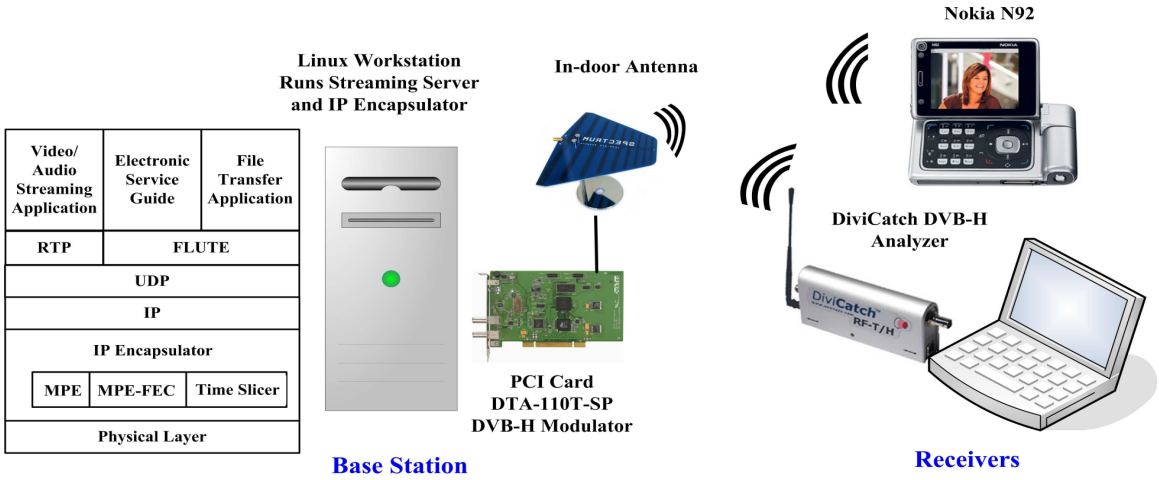


Figure 3.7: Setup of the Mobile TV (DVB-H) testbed.

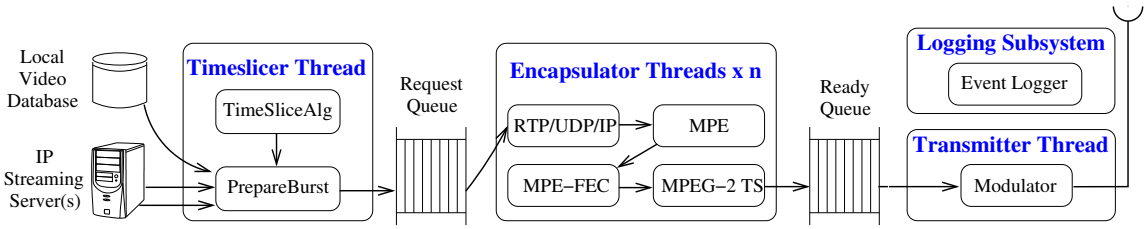


Figure 3.8: The proposed design for mobile TV base stations.

3.6.1 Base Station

We implement the base station in a Linux box in which we install the Radio Frequency (RF) signal modulator DTA-110T-SP available from DekTec [89]. This modulator implements the physical layer of the protocol stack and transmits DVB-H standard compliant signals via an indoor antenna. We use the low-cost antenna LP49-DTV [90]. The RF output level of the modulator, however, is quite low ($\sim$ -29 dBm) and can only reach up to 1-meter broadcast range for receivers with 6 dB gain antenna. Mobile devices typically have antenna gains much lower than 6 dB. We use the low-power amplifier available from Enensys [91] to boost the signal to about 0 dBm, which gives us approximately 20-meter range for cellular phones in our lab environment.

A simplified view of our software architecture of the base station is shown in Figure 3.8. Notice that this software architecture represents only the data plane of the testbed. The control plane is described in Section 3.6.3. In addition to this software architecture, we have developed a graphical user interface for managing the testbed; a snapshot of this interface is shown in Figure 3.9. Using this interface, many parameters of the base station can easily be configured. The parameters include: choosing the time slicing algorithm, setting the MPE-FEC frame size, adding and removing TV channels, choosing the physical layer modulation scheme and bitrate, setting the channel PIDs (packet identifiers) and IP multicast addresses, and so on. This interface facilitates conducting various experiments, because it does not require researchers to modify and compile the source code of different components of the system.

We now briefly describe the software architecture in Figure 3.8. The base station performs many operations in real time. Some of these operations involve blocking I/O tasks (e.g., reading data from a network socket) and expensive processing tasks (e.g., computing checksums and/or FEC codes of bursts). To efficiently support these operations, we employ

a multi-threaded design for the base station. As shown in Figure 3.8, our design has three thread groups: Timeslicer, Encapsulator, and Transmitter. Each of the first and third group has a single thread, while the second group has a pool of n threads, where n is a configuration parameter. We use this multi-threaded design to utilize the multi-core processor architectures that are quite common nowadays to provide the needed capacity to broadcast multiple TV channels. In addition, by adjusting thread priorities, the multi-threaded design with either single or multi-core processors can meet the real-time nature of broadcasting TV channels on a commodity PC or low-end server. For example, we make the priority of the Transmitter thread higher than other threads such that bursts are broadcast on time.

All threads process video data contained in *bursts*. A burst is a data structure that has fields for all needed protocol headers as well as a payload section. Every burst is treated as an independent unit, which allows us to move it from one processing stage to the next without copying data in the memory; only pointers are manipulated and passed between stages. The burst structure has fields for the RTP, UDP, IP, MPE, MPE-FEC, and MPEG-2 TS protocols in order to accelerate the encapsulation process. In addition, each burst has a timestamp field, which specifies the scheduled broadcast time for that burst. As shown in Figure 3.8, once a burst finishes processing in a stage, it is put in a priority queue for the next stage. The queue is sorted based on the timestamp in order to process bursts with closer deadlines earlier.

Timeslicer Thread. The Timeslicer thread receives video data from a local video database and/or remote IP video streaming servers. The local database may have pre-encoded TV programs such as TV series, documentary, and movies. The streaming servers can provide live content such as sports events and live talk shows. They also could provide pre-encoded content from online video databases. The Timeslicer thread has two main functions: TimeSliceAlg and PrepareBurst.

The TimeSliceAlg function implements the burst transmission scheduling algorithm, which determines the start time and size of each burst for every TV channel. The scheduling algorithm must ensure smooth playback of the video data by the mobile receivers, i.e., the receivers should not have buffer underflow or overflow instances. We designed the TimeSlicer thread to allow different scheduling algorithms to be easily implemented and evaluated. The PrepareBurst function takes the burst schedule produced by the TimeSliceAlg, and divides the data streams of different TV channels into bursts with appropriate sizes and timestamps. The output of the PrepareBurst function is bursts with filled payload sections

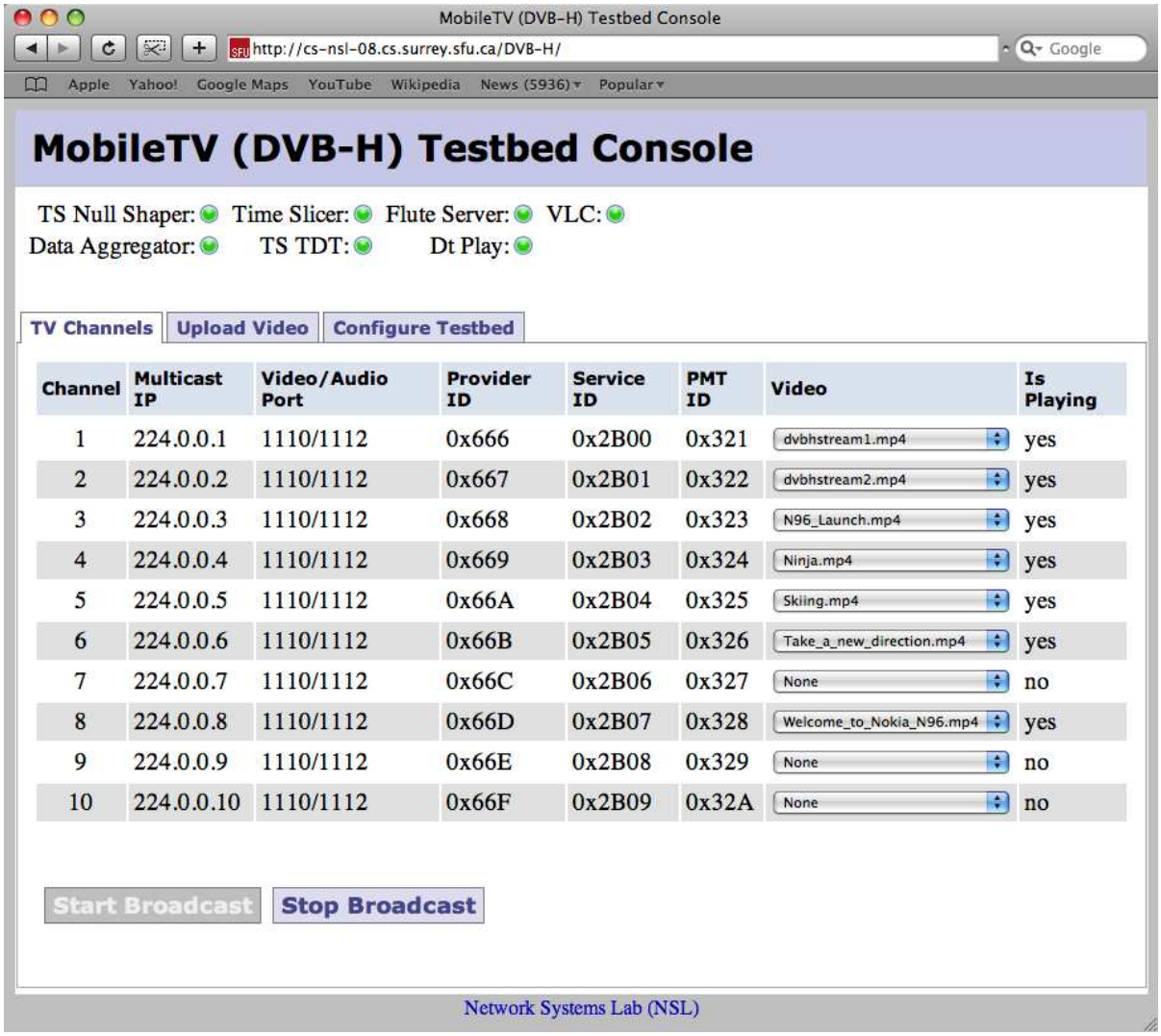


Figure 3.9: A snapshot of the web-based interface for managing the mobile TV testbed.

and timestamps as well as empty header fields (to be completed later by Encapsulator threads). Before filling the payload section, the PrepareBurst function performs a sanity check on the schedule to rule out any possible buffer overruns, due to the limitation on maximal MPE frame size. Note that, performing this check in the PrepareBurst simplifies the design: we do not re-implement this check in every TimeSliceAlg instance.

Encapsulator Threads. The Encapsulator threads complete the header fields for the different protocols according to the DVB-H standard, as shown in Figure 3.8. One of the functions performed by the Encapsulator threads is computing checksums and FEC codes for bursts, which is relatively expensive given that it needs to be done in real time for potentially many bursts. It is why we create a pool of Encapsulator threads to utilize any idle processing unit in the base station. We also separate the Encapsulation threads from the TimeSlicer thread because the latter could block on reading data from the local data base or from the network socket. Thus, the Encapsulator threads never block unless there is no burst in the Request Queue to be encapsulated. We set the priority of the Encapsulator threads to be lower than the priority of the TimeSlicer thread, such that once the TimeSlicer thread is ready to create bursts it will get a chance to do so.

Transmitter Thread. The Transmitter thread is assigned the highest priority among all other threads in our design in order to ensure the on-time delivery of bursts. It reads the bursts from the Ready Queue based on their timestamps and calls the appropriate function in the APIs of the RF signal modulator card. It also monitors the actual transmission of bursts and reports any abnormal delays. The Transmitter thread also supports time multiplexing of DVB-H bursts with non DVB-H services, such as PSI/SI (Program Specific Information/Service Information) information. This multiplexing is done as follows. During burst scheduling, the TimeSliceAlg considers the available channel bandwidth as $b - x$ kbps, where b is the total channel bandwidth and x is the amount of bandwidth assigned to non DVB-H services. b is a function of physical- and link-layer settings, while x is a configurable system parameter. Since the TimeSliceAlg schedules bursts for a channel bandwidth lower than the actual one, the resulting schedule consists of *reserved* time slots for carrying non DVB-H services at bit rate x. Therefore, the Transmitter thread can insert packets of non DVB-H services without affecting DVB-H services. Last, we note that DVB-H provides a constant bit rate channel: the base station *must* send traffic packets even when there is no data to transmit. To achieve this, the Transmitter thread sends null packets between any two bursts and for any unused bandwidth reserved for non DVB-H services. We intentionally

delay the null packet insertion to the Transmitter in order to reduce the processing overhead of other components such as Encapsulator threads: a statically allocated null packet is repeatedly reused without re-encapsulation.

Logging Subsystem: We have implemented an event logging module for analyzing the correctness and performance of the base station. The logging module supports multiple levels of events, which allows researcher to collect logs with the most appropriate amount of details. For example, to validate the implementation, we can enable DEBUG-level logs, which consist of many information including packet dumps. However, once the system is stable, we can switch to STATS-level logging which suppresses all DEBUG info but still saves statistical samples. Finally, for real deployments, the logging module can be disabled to avoid any processing and memory overhead.

3.6.2 Mobile Receivers and Data Analyzers

The goal of the proposed testbed is to analyze various quantitative as well as qualitative performance metrics of mobile TV networks. These metrics include visual quality, energy consumption of mobile devices, channel switching delay, user interactivity with the system, and ease and intuitiveness of the electronic service guide. To assess these performance metrics, we have integrated different types of receivers into the proposed testbed, which are briefly described below.

Nokia N96 Mobile Phones. We use Nokia N96 (and its predecessor N92) phones as receivers. These phones are equipped with a DVB-H signal receiver, the receiver-side part of the DVB-H protocol and a video player. These phones are used to verify the correctness and compliance of our base station implementation to the DVB-H and IP Datacast standards. The phones can also be used in user studies to assess the subjective visual quality of the broadcast videos. This can be useful, for example, when new video encoders are being tested or when searching for the bit rates for different types of video content to maximize the visual quality on actual mobile receivers. Also mobile phones can be used to evaluate the user interface of the mobile services and test new applications such as integrating cell phone and video broadcasting services (e.g., interactive TV shows with user inputs/votes).

At lower layers, the actual energy consumption of receiving DVB-H signals and switching delay among TV channels can be measured on the Nokia phone. The energy consumption can be measured using the Nokia Energy Profiler application, which runs on the mobile phone to monitor energy consumption in real time. The Energy Profiler is quite flexible,

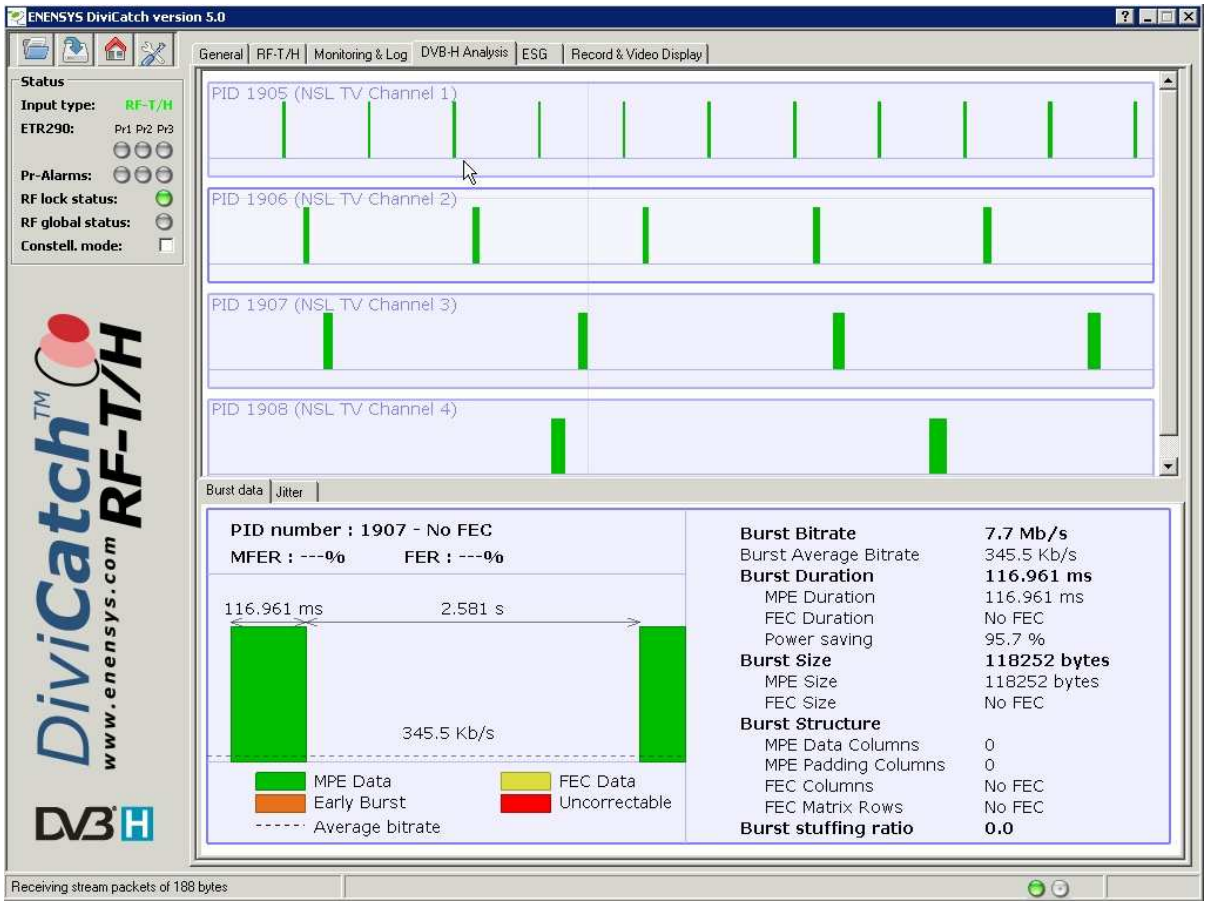


Figure 3.10: A snapshot of the graphical user interface of the DiviCatch analyzer.

and it supports exporting measurement data logs to files for further analysis. The channel switching delay, on the other hand, can be measured either by instrumenting the video player or by implementing a utility as the middle-box between the DVB-H receiver and the video player. The second approach is more feasible because the video player on the Nokia phone is proprietary, which prevents us from augmenting it.

DVB-H Signal Analyzers. As the mobile TV application is proprietary, the mobile phones cannot be used for evaluating the correctness and performance of different protocols in the testbed implementation. To cope with this, we add a DVB-H analyzer, called DiviCatch RF T/H tester, available from Enensys [92] to the testbed. This analyzer is attached to a PC via a USB port, and comes with a visualization software that runs on Windows. The software provides detailed real-time information on the RF signals, the MPE frames, the burst schedules, the burst jitters, among others. Figure 3.10 presents a snapshot of this It also comes with visualization software, which presents crucial information in an intuitive and user-friendly way, and is suitable for field testing, e.g., to locate dead zones in a real deployment. However, based on our experience, we believe that this analyzer is not very useful in validating the testbed implementation. This is because the analyzer, as a real time tester, does not allow byte-level packet analysis. Hence, it can only diagnose serious implementation errors such as early bursts.

To overcome this limitation, we use another DVB-H analyzer, called dvbSAM, available from Decontis [93]. Different from DiviCatch, dvbSAM is a software based solution that is compatible with many DVB-T USB receivers in the market. For example, we use a low-cost WinTV-NOVA-T receiver from Hauppauge [94]. dvbSAM is also a Windows application. It allows us to dump various information, such as PSI/SI tables and ESG fragments, in hexadecimal format to debug the testbed implementation. Hence, we believe dvbSAM is more useful for researchers to validate the algorithms and protocols implemented in the base station, while DiviCatch is suitable for planning broadcast networks. We note that the costs of these two analyzers are comparable: $5,000–$6,000.

Our Open-Source Analyzer. The above commercial analyzers are useful for debugging the implementation of new protocols and algorithms. However, their source codes are not available, and therefore cannot be programmed to measure new performance metrics that may be of interest to researchers. For example, while measuring the channel switching delay is quite important [29, 33, 34], none of the commercial analyzers provides a systematic way to collect samples of the channel switching delay. In addition, they are relatively

expensive. Hence, an open-source DVB-H analyzer is desirable.

We have developed an open-source analyzer in Java that runs on Linux. The analyzer leverages several utilities developed by the Linux TV project [95]. The Linux TV project, part of the official Linux 2.6 kernel, supports many DVB-T TV receivers, including the low-cost WinTV-NOVA-T receiver [94]. Our analyzer supports capturing TS and IP streams as follows. Upon setting up the DVB drivers, the USB receiver shows up as /dev/dvb, which allows us to perform a signal scan using the dvbscan utility. The dvbscan utility returns the modulator parameters, such as frequency, bandwidth, and modulation scheme. With these parameters, the USB receiver can be tuned to the right network using the tzap utility. Then, we can use the dvbsnoop utility to record and analyze the MPEG-2 TS streams. To extract IP streams from the TS streams, the analyzer uses the dvbnet utility to configure a network interface for each DVB-H channel, and demultiplex TS packets based on the PIDs. The analyzer then uses libpcap library to capture the IP streams.

With the capability of capturing both TS and IP streams, our analyzer can measure system performance from different aspects, such as the energy saving and the channel switching delay. Our analyzer has four parts: tuner, channel monitor, video player, and user interface. The tuner sets up the DVB-T receiver and network interfaces. Each channel monitor is a thread capturing IP packets for a specific DVB-H channel, where each packet is associated with a receiving timestamp. Based on the timestamps, the channel monitor clusters packets into bursts, then measures the performance metrics. At any moment, a channel monitor can relay the captured packets to the video player for play-out. The user interface presents the statistics collected by the channel monitors, and allow users to select a channel to watch. Since the analyzer is open-source, it can be easily extended to evaluate other performance metrics.

3.6.3 Signaling and Electronic Service Guide

This section describes the control plane of the testbed. Unlike the data plane which handles broadcasting the actual data of TV channels, the control plane manages the auxiliary information transmitted by the base station in order to enable mobile devices to receive and successfully decode different TV channels. The control plane has two parts: PSI/SI (Program Service Information and Service Information) and ESG (Electronic Service Guide). We describe each of them in the following.

The PSI/SI provides link-layer signaling for carrying network configurations. It is organized as tables, where each table is encapsulated in one or more TS packets. The TS packets of the same PSI/SI table share a PID, which allows receivers to reassemble PSI/SI tables. Several tables are defined in the standard, e.g., Network Information Table (NIT) carries the tuning information such as frequency, bandwidth, and modulation scheme. To enable new receivers to find the DVB-H services, every PSI/SI table is re-transmitted every few hundred milliseconds (the standard specifies various maximum retransmission period for different tables). As aforementioned, the TS packets of PSI/SI tables are multiplexed with the DVB-H packets by the Transmitter thread. This can be done very efficiently by repeatedly sending the same packets in memory, as most PSI/SI tables are fairly stable over time.

In the testbed, we abstract each PSI/SI table as a class that provides three functions. First, it allows users to modify table content on-line, through a graphic user interface. Second, it can read/write the content from/to a human-readable file, which enables users to change the content offline. Third, and most importantly, it can pack table content into TS packets following the packet format defined in the standard. The packed TS packets are then sent by the Transmitter thread.

The ESG provides application-layer signaling to describe the offered TV services, which allows mobile devices to present information about available programs to users. In addition, ESG also carries session information that enables mobile devices to locate and play the specific IP streams. ESG information is written as XML files, which are then divided into fragments. There are several fragment types defined in the standard, e.g., Content fragments provide textual description of TV programs. To reduce the processing time at receivers, ESG fragments are encapsulated into a few binary container files with indexing fields. These files are sent to mobile devices using the File Delivery over Unidirectional Transport (FLUTE) protocol. To support new receivers, ESG fragments are also periodically retransmitted, but with a longer period (in the order of seconds). The IP streams of the FLUTE protocol are encapsulated as DVB-H channels, which are then time-sliced with other DVB-H TV channels.

We implemented ESG as XML files in the testbed. We developed scripts to systematically create these XML files, and encapsulate them into binary container files. To send these container files, we integrated an open-source FLUTE implementation [96] into our

testbed after several modifications/customizations. The customizations are required in order to comply with the IP Datacast standard [10], because the FLUTE implementation targets many applications other than DVB-H ESG. We should mention that our experience indicates that dvbSAM analyzer is better than DiviCatch for debugging ESG: dvbSAM provides more details on ESG fragments and is more stable.

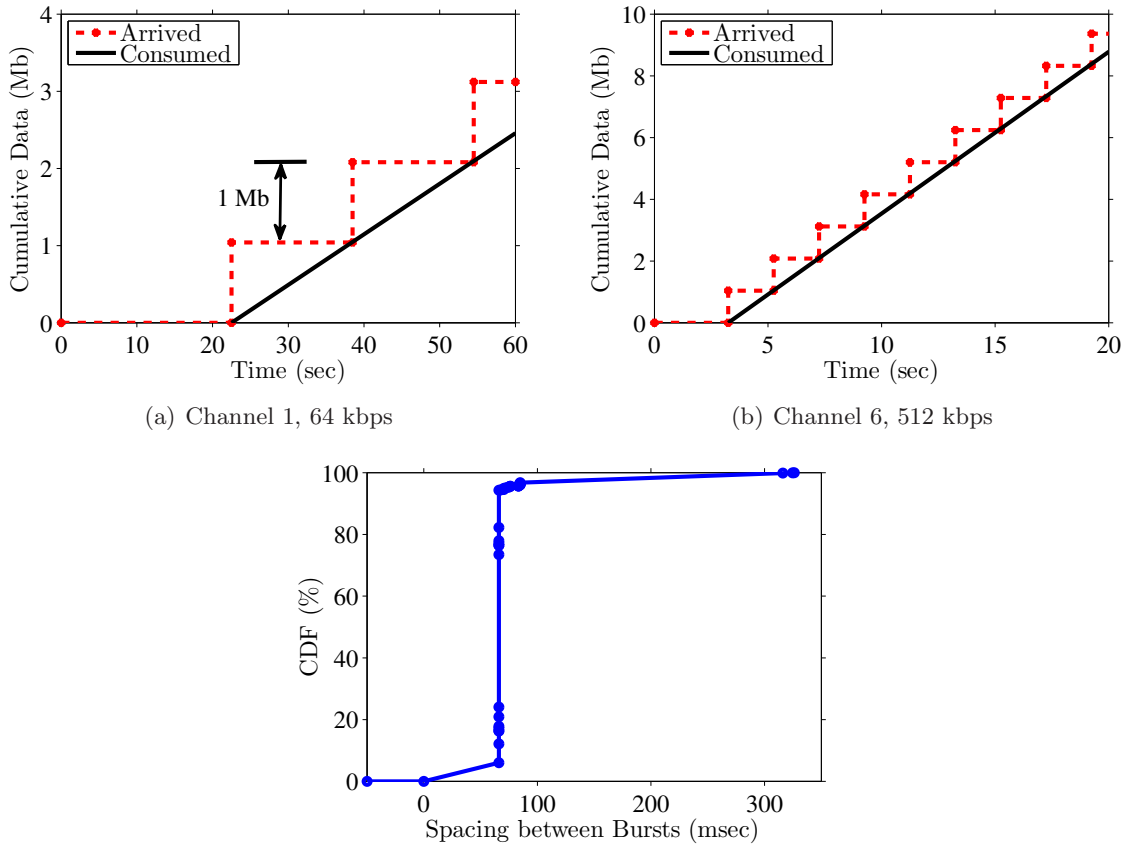
3.7 Evaluation of the Optimal P2OPT Algorithm

We evaluate the proposed P2OPT algorithm using actual implementation in the mobile TV testbed (described in Section 3.6) as well as simulation. We also analyze the limitations of the current practice of assigning the same bit rate for all TV channels, and we experimentally show that our power of two bit rate increments solution can result in better viewing experience by reducing quality variation among all TV channels.

3.7.1 Setup of the Mobile TV Testbed

As illustrated in Section 3.6, we have set up a testbed for DVB-H networks, which provides a realistic platform for analyzing our burst scheduling algorithm. We configure the modulator to use a 5 MHz radio channel with Quadrature Phase-Shift Keying (QPSK) modulation. This leads to 5.445 Mbps air medium bandwidth according to the DVB-H standard [16]. We concurrently broadcast 9 TV channels using our P2OPT algorithm for 10 minutes. These TV channels are classified into 4 classes: 2 channels at 64 kbps, 3 channels at 256 kbps, 2 channels at 512 kbps, and 2 channels at 1024 kbps. The receiver buffer size is 1 Mb. For each of these TV channels, we set up a streaming server on the base station to send 1-kB IP packets at the specified bit rate. To conduct statistically meaningful performance analysis, we collect detailed event logs from the base station. The logs contain the start and end times (in msec) of broadcasting every burst of data and its size.

We develop software utilities to analyze the logs for three performance metrics: bit rate, time spacing between successive bursts, and energy saving. We compute the bit rate for each TV channel by considering the start times of two consecutive bursts and the burst size. We use the bit rate to verify that our burst scheduling algorithm leads to no buffer under/overflow instances. We compute the time spacing between bursts by first sorting bursts of all TV channels 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,



(a) Channel 1, 64 kbps

(b) Channel 6, 512 kbps

(c) Bursts of all channels

Figure 3.11: Experimental validation of the P2OPT algorithm: (a) and (b) show no over/underflow instances, and (c) shows no burst conflicts.

burst. We use the time spacing to verify that there are no burst conflicts, as a positive time spacing indicates bursts do not intersect with each other. We compute the energy saving for each TV channel as the ratio between the receiving circuit on time and off time. We assume the overhead duration $T_o = 100$ msec.

3.7.2 Experimental Results

Experimental validation of P2OPT correctness. We first validate the correctness of our P2OPT algorithm from the actual testbed implementation. Figure 3.11 demonstrates the correctness of the P2OPT algorithm. In Figs. 3.11(a) and 3.11(b), we plot the cumulative data received by receivers of two sample channels as the time progresses (other results are similar). We also show the consumed data with the time. To account for the worst case, we assume that the receiver starts consuming (playing back) the video data immediately after receiving a burst. The two figures clearly show that there are: (i) no buffer underflow instances as the consumption line never crosses the stair-case curve representing arrived data, and (ii) no buffer overflow instances as the distance between the data arrival and consumption curves never exceeds the buffer size (1 Mb). Notice that Figs. 3.11(a) and 3.11(b) show only short time periods for clarity; but since these short periods cover multiple scheduling windows and the burst scheduling is identical in successive frames, the results are the same for the whole streaming period.

In order to show that there are no burst conflicts, we plot the CDF curve of the time spacing between successive bursts in Figure 3.11(c). This CDF curve is computed from all bursts of all TV channels broadcast during the experiment period (10 minutes). Negative time spacing would indicate that two bursts are intersecting in time, i.e., burst conflict. This figure clearly shows that our P2OPT algorithm results in no burst conflicts.

Energy saving achieved by P2OPT. Next, we report the energy saving achieved by our algorithm. Figure 3.12 presents the energy saving of each TV channel. We observe that the energy saving for low bit rate TV channels can be as high as 99%, while it is only 76% for high bit rate TV channels. This dramatic difference emphasizes the importance of broadcasting TV channels at heterogeneous bit rates: to maximize energy saving on mobile devices, a TV channel should be sent at the lowest bit rate that fulfills its minimum quality requirement.

Optimality of P2OPT. Last, we verify that the energy saving achieved by our P2OPT algorithm is indeed optimal. To do this, we compute the absolute maximum energy saving

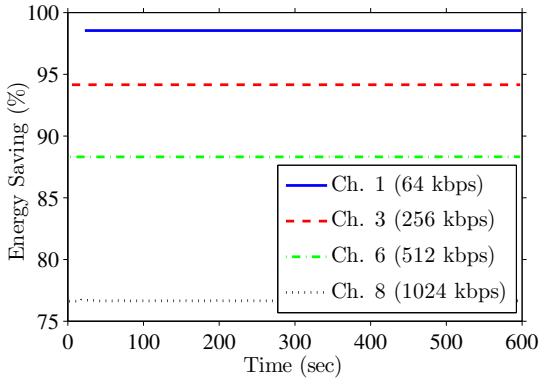


Figure 3.12: Energy saving achieved by our P2OPT algorithm for individual TV channels.

that can be achieved by any algorithm for a given TV channel. We compute this maximum by making the base station broadcast only the given TV channel. In this case, the base station easily maximizes the energy saving by allocating the largest burst that can fill the receiver's buffer. The receiving circuit of the receiver is then turned off till the data of this burst is consumed. We repeat this experiment nine times; once for each considered TV channel, and we compute the maximum possible energy saving. We then run our algorithm to compute the burst schedule for the nine TV channels, and we make the base station broadcast all of them *concurrently*. We compute the energy saving for each TV channel. Sample results for channels 1 and 6 are presented in Figure 3.13; all other results are similar. The figure confirms the optimality of the P2OPT algorithm in terms of energy saving.

3.7.3 Simulation Analysis

To evaluate our algorithm under wider ranges of parameters, we have implemented a simulator for mobile TV networks. The simulator captures the important aspects of mobile TV networks that are relevant to the burst scheduling problem, and it abstracts away details such as sending program guide to mobile devices and FEC protection on video packets, which are orthogonal to burst scheduling algorithms.

We simulate a mobile TV network with a wireless medium bandwidth $R = 6.4$ Mbps. We set the receiver buffer size $b = 2$ Mb and the overhead duration $T_o = 100$ msec. We broadcast multiple TV channels using the optimal burst schedules computed by P2OPT. We vary the number of TV channels to achieve different target bandwidth utilization values. We

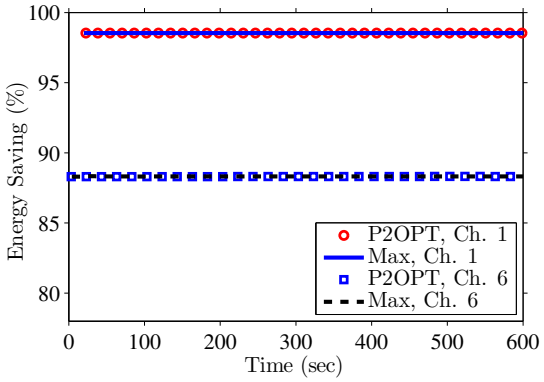


Figure 3.13: Optimality of P2OPT: Comparing the energy saving achieved by P2OPT against the absolute maximum saving.

consider bandwidth utilization from 30% to 100% to cover most practical scenarios, and to validate the scalability of our P2OPT algorithm. We randomly construct burst scheduling problems for each bandwidth utilization value by selecting TV channel bit rates from 50, 100, 200, 400, and 800 kbps, so that the total bit rate does not exceed the bandwidth utilization value. We solve the burst scheduling problems using our P2OPT algorithm and measure the following metrics: maximum energy saving, average channel switching delay, and running time. We repeat each experiment 100 times and report the minimum, mean, and maximum values of each performance metric. We run our simulation on a commodity PC running Linux.

Figure 3.14 summarizes the performance of the P2OPT algorithm. Figure 3.14(a) shows that our algorithm constantly leads to high energy saving: more than 92%. This means that mobile devices can turn off their receiving circuits for 92% of the time, which increases battery life and watch time. Figure 3.14(b) implies that the channel switching delay is less than 5 seconds on average. Figure 3.14(c) reports the running time of our P2OPT algorithm. This figure shows that our algorithm is very efficient: it terminates in less 30 msec on average under all considered bandwidth utilization levels. Most importantly, Figure 3.14(c) implies that our P2OPT algorithm is scalable and can be employed in fully loaded mobile TV networks.

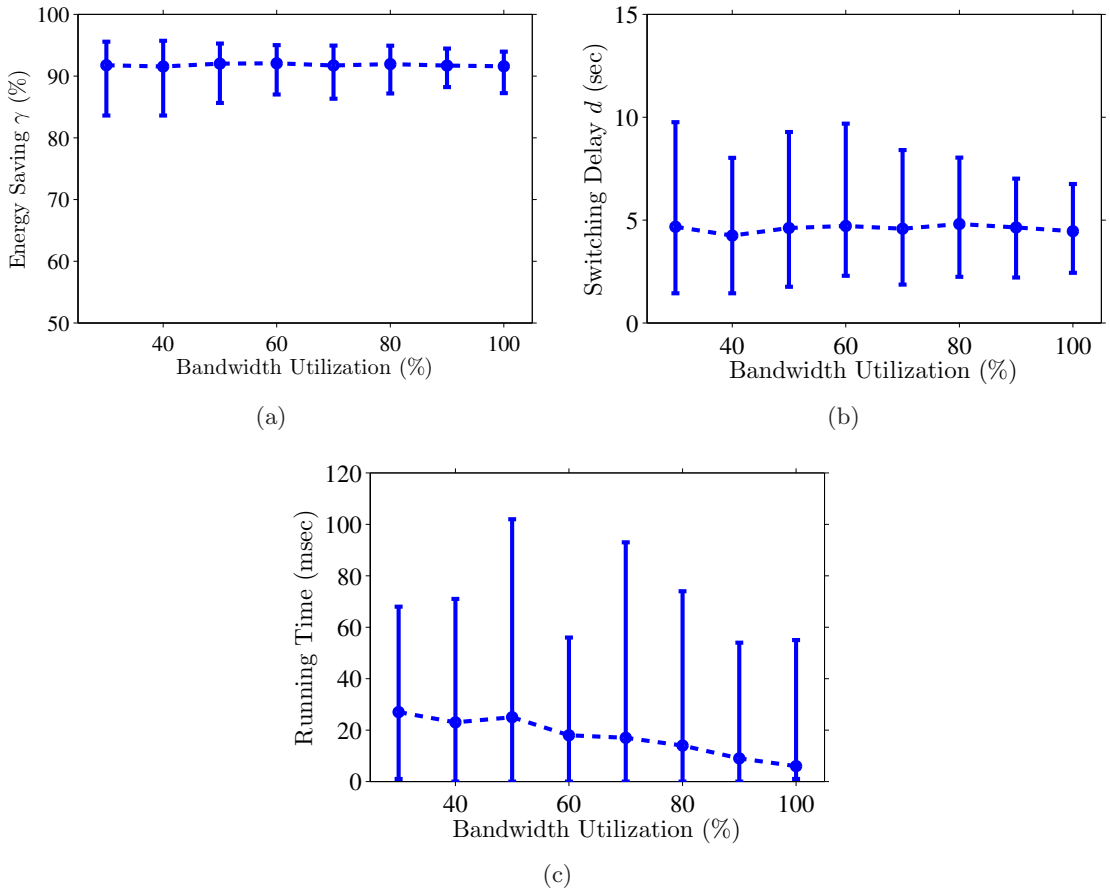


Figure 3.14: Performance of the P2OPT algorithm: (a) energy saving, (b) channel switching delay, and (c) running time.

Table 3.2: List of video sequences for experiments.

Seq.	**Res.**	**FPS**	**Seq.**	**Res.**	**FPS**
Foreman	QCIF	7.5	Mobile	QCIF	7.5
Foreman	QCIF	15	Mobile	QCIF	15
Foreman	CIF	15	Mobile	CIF	15
Foreman	CIF	30	Mobile	CIF	30
Soccer	4CIF	30	Soccer	4CIF	60

3.7.4 Power of Two versus Uniform Bit Rates

We experimentally quantify the potential benefits of classifying TV channels into multiple classes with heterogeneous bit rates. Sending all TV channels at the same bit rate can lead to under utilization of the wireless medium and/or degraded video quality. This is because different video streams need to be encoded at bit rates proportional to their content complexity. For example, to achieve an acceptable video quality, encoding a sports event, such as a football game, requires higher bit rate than encoding a talk show. In addition to content complexity, video frame rates and display dimensions also have significant impact on the TV channel bit rates for achieving a target video quality.

To verify the above intuition and quantify its impacts, we encode three video sequences: Foreman, Mobile, and Soccer at five different frame rates and three screen resolutions. Foreman has a talking person with camera movements, Mobile contains many spatial details, and Soccer features fast motion. Table 3.2 lists all considered, 10-sec long, video sequences. We encode each of these sequences into a video stream using the H.264 reference coder [97] with typical configurations used in previous works [98, 99]. To derive the R-D curves, we encode each video sequence at six sampling bit rates: from 32 to 1024 kbps. We then decode each coded stream and compute the average video quality quantified by the peak signal-to-noise ratio (PSNR). Figure 3.15 shows four sample R-D curves among all considered video sequences. This figure indicates that encoding all video sequences at a uniform bit rate results in significant quality variation. For example, at 512 kbps, the quality difference among different sequences can be as high as 20 dB. Serving all TV channels at a uniform bit rate, therefore, leads to huge quality variations among channels, which degrades user experience especially when they switch channels.

To show that proper selection of bit rates can reduce the quality variation, we consider the following rate allocation problem for multiple concurrent TV channels: given the R-D characteristic of each TV channel, determine the best power of two coding rates for individual TV channels that minimize the video quality variation. In particular, we consider the ten video sequences listed in Table 3.2 and the six coding rates used before, and compute the best rate allocation using exhaustive search. We then compare the resulting quality variation against the quality variations if we were broadcasting all TV channels at any of the uniform bit rates. We note that we did not solve this rate allocation problem with more efficient algorithms because we only want to quantify the potential benefits of using our

burst scheduling algorithm.

Figure 3.16 reports the quality variation produced by the optimal power of two and uniform bit rate allocations. The quality variation is computed as the standard deviation of the PSNR values of the ten video sequences when they are encoded at the specified bit rate. This figure shows that broadcasting TV channels at uniform bit rates can lead to high quality variations, up to almost 10 dB in terms of standard deviation. In contrast, broadcasting TV channels at power of two bit rates reduces video quality variation: a standard deviation less than 1 dB can be achieved with only six possible encoding rates. As low video quality variation is desirable, service providers who use our P2OPT algorithm to broadcast TV channels at heterogeneous bit rates can provide better service quality and higher user satisfaction, which in turn will increase their revenue.

Finally, we study advantages of using the P2OPT algorithm other than lower quality variation. We consider a network operator who needs to broadcast the aforementioned ten video sequences at video quality no less than the basic quality of 30 dB in PSNR. If uniform encoding rate is employed, all video sequences are encoded at 1 Mbps to achieve this basic quality. In contrast, using the P2OPT algorithm, video sequences with fewer details/motions can be encoded at as low as 32 kbps while still achieving the basic quality. Consider a mobile TV network with a wireless medium bandwidth $R = 10.556$ Mbps, the network load with uniform encoding rate is $1024 * 10/10556 = 94.73\%$, and the load with P2OPT algorithm is $4096/1056 = 38.50\%$. Clearly, using P2OPT algorithm enables network operators to reduce the network load and increase the number of concurrently broadcast TV channels. Next, we plot the energy saving of individual TV channels in Figure 3.17. This figure shows that the P2OPT algorithm allows mobile devices to save more energy. More precisely, broadcasting the video sequences at a uniform encoding rate results in 80.30% energy saving for all TV channels, while broadcasting them using the P2OPT algorithm leads to 92.18% energy saving on average. In summary, in addition to lower quality variation, the P2OPT algorithm enables network operators to broadcast more TV channels, and allows mobile devices to save more energy.

3.8 Near-Optimal Solution for the General Problem

We solve the general form formulation of Problem 1 in this section. We first observe that the formulation in Eq. (3.1) has two *tightly-coupled* constraints: (i) no burst intersection

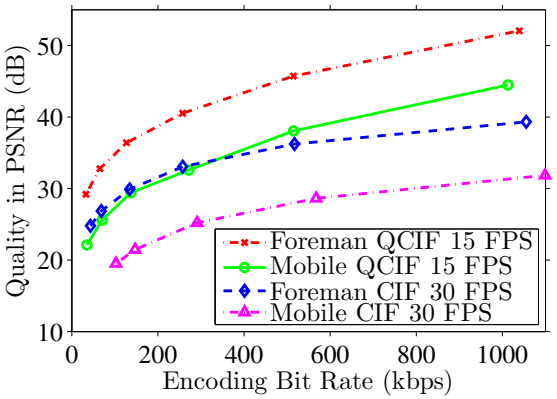


Figure 3.15: Sample R-D curves for considered video sequences.

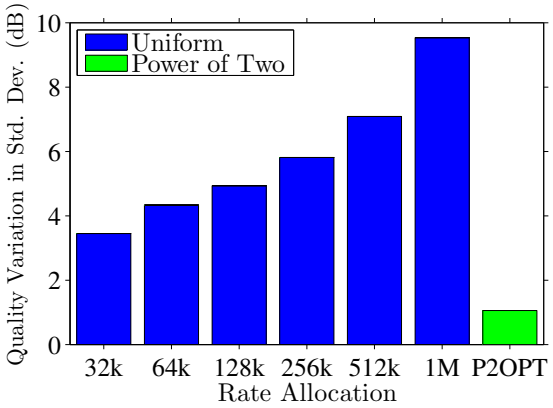


Figure 3.16: Comparison of quality variation for uniform and power of two bit rates.

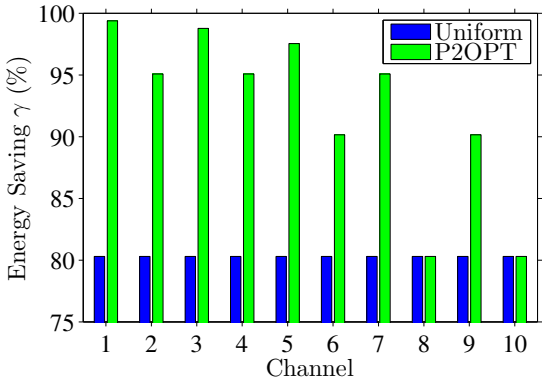


Figure 3.17: Comparison of the energy saving achieved by uniform and P2OPT.

(Eq. (3.1b)) and no receiver buffer violation (Eqs. (3.1c)–(3.1f)). These two constraints prevent us from employing many common techniques for solving machine scheduling problems. Therefore, in Section 3.8.1, we propose to *decouple* these two constraints by transforming the *original* formulation in Eq. (3.1) into another formulation, which only has *one* constraint and is easier to solve. We call the new formulation *transformed* formulation. In Section 3.8.2, we present an efficient algorithm to solve the transformed formulation with near-optimality. We also convert the near-optimal schedule for the transformed formulation back to the original formulation in the same subsection. In Section 3.8.3, we analytically study the proposed algorithm, and we show its correctness, complexity, and approximation factor.

3.8.1 Transform

The goal of the transform is to construct a formulation with a single constraint of no burst collisions. The transformed formulation is then similar to those of the machine scheduling problems, and can be efficiently solved with near-optimality. To achieve this goal, we propose to split the receiver buffer into two equal-sized buffers, called B and B', and we divide the scheduling window into multiple subwindows, such that the number of bits that is received for smooth playout in each subwindow never exceeds the size of B (or B'). In every subwindow, a mobile device stores the received data in one of its buffer, say B, for later usage, and decodes the previously received data from another buffer B'. The mobile device swaps its two buffers upon a new subwindow is reached: it stores the received data in the empty buffer B', and decodes the data from the filled buffer B. For smooth playouts, we need to make sure that the number of received bits in each subwindow is the same as the number of consumed bits in the subsequent subwindow, and we present a systematic way of doing this in the following.

We let $\mathbf{w}_s$ be a set of subwindows of video stream s, where $s = 1, 2, \ldots, S$. More

specifically, we define $\mathbf{w}_s = \{w_s^k \mid k = 1, 2, \ldots, \lceil 2pr_s/Q \rceil\}$ and $w_s^k = <x_s^k, y_s^k, z_s^k>$, where:

$$x_s^k = (k-1)Q/(2r_s); \tag{3.4a}$$

$$z_s^k = \begin{cases} kQ/(2r_s), \;\; 1 \le k \le \lfloor 2pr_s/Q \rfloor; \\ p, \;\; k = \lceil 2pr_s/Q \rceil \text{ if } 2pr_s/Q \notin \mathbb{Z}; \end{cases} \tag{3.4b}$$

$$y_s^k = \begin{cases} Q/(2R), \;\; 1 \le k \le \lfloor 2pr_s/Q \rfloor; \\ \frac{Q}{2R} \times \frac{p \bmod \frac{Q}{2r_s}}{\frac{Q}{2r_s}}, \;\; k = \lceil \frac{2pr_s}{Q} \rceil \text{ if } \frac{2pr_s}{Q} \notin \mathbb{Z}. \end{cases} \tag{3.4c}$$

In this definition, w_s^k is subwindow k of video stream s, and is written as a 3-tuple $<x_s^k, y_s^k, z_s^k>$, where x_s^k and z_s^k are the start and end times of this subwindow, and y_s^k is the total burst time that should be assigned to the video stream between x_s^k and z_s^k for smooth playouts. With the above notations, we write the transformed formulation as:

$$\min_{\mathbf{L}} \quad \sum_{s=1}^{S} n_s \tag{3.5a}$$

$$\text{s.t.} \quad \left[f_s^{\hat{k}}, f_s^{\hat{k}} + \tfrac{b_s^{\hat{k}}}{R}\right) \bigcap \left[f_{\bar{s}}^{\bar{k}}, f_{\bar{s}}^{\bar{k}} + \tfrac{b_{\bar{s}}^{\bar{k}}}{R}\right) = \varnothing; \tag{3.5b}$$

$$\sum_{x_s^k \le f_s^i \le z_s^k} b_s^i = R y_s^k; \tag{3.5c}$$

$$\forall\, 1 \le s \ne \bar{s} \le S,\; 1 \le k \le \lceil 2pr_s/Q \rceil, 1 \le \hat{k} \le n_s, 1 \le \bar{k} \le n_{\bar{s}}.$$

In this formulation, the objective function in Eq. (3.5a) minimizes the number of total bursts in the scheduling window. This is because fewer bursts lead to less overhead due to waking up receiving circuits, and thus result in higher energy saving. The constraint in Eq. (3.5b) prevents any intersected bursts, and the constraint in Eq. (3.5c) ensures that the total burst length scheduled for video stream s between times x_s^k and z_s^k *equals* to y_s^k. We notice that the rationale behind the definition of $\mathbf{w}_s$ is to make sure that every subwindow w_s^k gets scheduled bursts that are sufficient to transmit $Q/2$ kb video data (indicated by Eq. (3.4c)), which equals the amount of data required for smooth playouts in the subsequent subwindow (indicated by Eqs. (3.4a) and (3.4b)). In addition, while each subwindow w_s^k ($s = 1, 2, \ldots, S$ and $k = 1, 2, \ldots, \lceil 2pr_s/Q \rceil$) requires total burst time y_s^k, it needs *not* be from a *single* burst.

3.8.2 Efficient Algorithm

We present an efficient algorithm to solve the transformed formulation in Eq. (3.5) in the following, and we call it Double Buffering Scheduling (DBS) algorithm. We define *decision points* as the time instances at which either a new subwindow starts, i.e., at time x_s^k for any s and k, or bursts scheduled to a subwindow have met the required burst length, i.e., satisfying the constraint in Eq. (3.5c). At each decision point t, our scheduling algorithm schedules a burst for the subwindow with the smallest end time z_s^k among all outstanding subwindows with start time earlier than the current time, i.e., $x_s^k \leq t$. We say a subwindow is outstanding if and only if it requires more bursts, i.e., the current schedule has not satisfied its constraint in Eq. (3.5c) yet. Moreover, we let e_s^k be the completion time of subwindow w_s^k, where e_s^k represents the time at which its constraint in Eq. (3.5c) is met. Our scheduling algorithm terminates whenever there is no outstanding subwindow nor a subwindow that has a start time in the future. In Lemma 1, we prove that our algorithm gives a burst schedule that minimizes the subwindow lateness, which is defined as $e_s^k - z_s^k$. The achieved lateness allows us to determine whether the resulting burst schedule is feasible or not. More precisely, we check whether $e_s^k - z_s^k$ is non-positive for all s and k. If this is true, we know that the resulting burst scheduling is feasible. Otherwise, our algorithm prompts the network operator to reduce the number of video streams, because the required bandwidth exceeds the network capacity.

Figure 3.18 gives a high-level pseudo code of the proposed DBS burst scheduling algorithm. In lines 1–4, the algorithm constructs the transformed formulation by defining the subwindows for all video streams. In lines 5–9, it traverses through all decision points in the transformed formulation and builds a burst schedule. Last, it checks whether the resulting schedule is feasible in lines 10–13.

3.8.3 Analysis

We first show, in the next lemma, that the DBS algorithm can solve the transformed formulation.

Lemma 1. *The DBS algorithm finds a burst schedule for the transformed formulation in Eq. (3.5) if and only if one exists.*

Proof. First, we prove that if the DBS algorithm returns a schedule $\mathbf{L}$, then $\mathbf{L}$ is a feasible solution for the formulation in Eq. (3.5). In Figure 3.18, the for-loop between lines

Double Buffering Scheduling (DBS) Algorithm

```
1.  // construct the transformed formulation
2.  for s = 1 to S
3.    for k = 1 to ⌈2pr_s/Q⌉
4.      determine x_s^k, y_s^k, and z_s^k using Eqs. (3.4a)—(3.4c)
5.  // schedule bursts
6.  let L = ∅
7.  foreach decision point {
8.    add a burst from times t_c to t_n to channel s, where w_s^k has the smallest z_s^k among
8.    outstanding subwindows, t_c is current time, and t_n is time of the next decision point
9.  }
10. // feasibility check
11. if max{e_s^k − z_s^k} ≤ 0 // complete on time
12.   return L
13. return no feasible schedule
```

Figure 3.18: A near-optimal algorithm to solve the general form burst scheduling problem.

7–9 iterates through all decision points, which are defined as the time instances at which a subwindow starts or a subwindow reaches its required burst length. While at each decision point, line 8 schedules the outstanding subwindow with the smallest subwindow end time. Therefore, resulting schedules of the DBS algorithm have no burst collisions, and satisfies Eq. (3.5b). We notice that, this for-loop actually composes a burst schedule $\mathbf{L}$ that minimizes the maximal subwindow lateness $e_s^k - z_s^k$ among all subwindows of all video streams. This can be proved by transforming any optimal burst schedule to $\mathbf{L}$ with a finite number of burst swaps without compromising the schedule feasibility. This burst swapping technique is similar to the one used in [83, Theorem 4.4], which solves a single machine preemptive scheduling problem for minimizing task lateness. In lines 11–13, the algorithm checks whether $\mathbf{L}$ satisfies Eq. (3.5c) by comparing the maximal subwindow lateness against zero. Since $\mathbf{L}$ returned in line 13 satisfies Eq. (3.5), the proof follows.

Second, we show that if the algorithm does not find a feasible burst schedule, there exists no solution for the formulation in Eq. (3.5). We prove this by contradiction. Assume that there is a feasible schedule $\hat{\mathbf{L}}$ and our algorithm in line 13 claims the resulting $\mathbf{L}$ is not feasible. Note that line 13 says that $\mathbf{L}$ is not feasible only if the corresponding

maximal subwindow lateness is positive. However, the feasible schedule $\hat{\mathbf{L}}$ must have a non-positive maximal subwindow lateness, and the maximal subwindow lateness of $\hat{\mathbf{L}}$ is smaller than that of $\mathbf{L}$. However, as discussed above the for-loop between lines 7–9 minimizes the maximal subwindow lateness, which leads to a contradiction because the algorithm should have returned $\hat{\mathbf{L}}$ instead. □

We then show, in the next lemma, that the proposed transform does not affect the existence of feasible burst schedules.

Lemma 2. *There exists a feasible schedule for the transformed formulation in Eq. (3.5) if and only if there exists a feasible schedule for the original formulation in Eq. (3.1).*

Proof. We claim that for an arbitrary buffer size $\bar{Q}$, there exists a feasible burst schedule for the original burst scheduling formulation in Eq. (3.1) if and only if $\sum_{s=1}^{S} r_s \leq R$. We show this by constructing a feasible schedule as follows. Without loss of generality, we assume that $r_1 \leq r_2 \leq \cdots \leq r_S$. We allocate a burst to each video stream every $\bar{Q}/r_S$ sec, the burst size for video stream s is $\frac{\bar{Q}}{r_S} \times r_s$, and bursts for different video streams are placed in a round-robin fashion. We note that the resulting schedule leads to no buffer violations for any video stream s. More importantly, the total burst length in each round ($\sum_{s=1}^{S} \frac{\bar{Q}}{r_S} \times \frac{r_s}{R}$) is no longer than the round duration ($\bar{Q}/r_S$) if and only if $\sum_{s=1}^{S} r_s \leq R$. This shows that the existence of a feasible burst schedule for the original formulation in Eq. (3.1) is independent from the receiver buffer size Q. Hence the proposed transform does not affect the existence of feasible schedules. □

With these two lemmas, we can now show that the DBS algorithm always finds a feasible burst schedule for the original burst scheduling formulation in Eq. (3.1).

Theorem 4 (Correctness)**.** *The DBS algorithm produces a feasible schedule for the original formulation in Eq. (3.1).*

Proof. Lemmas 1 and 2 reduce the proof to showing that any feasible schedule for the formulation in Eq. (3.5) is also feasible for the formulation in Eq. (3.1). In lines 2–4, we divide each scheduling window into $\lceil 2pr_s/Q \rceil$ disjoint subwindows, where each subwindow has a required burst length $Q/2$ kb. Therefore, following the definition of y_s^k in Eq. (3.4c), any feasible burst schedule $\mathbf{L}$ for the transformed formulation in Eq. (3.5) transmits $Q/2$ kb data in every subwindow. Furthermore, the definitions of x_s^k and z_s^k in Eqs. (3.4a) and

(3.4b) indicate that the duration of each subwindow is $Q/(2r_s)$ and the amount of data to support smooth playout in a subwindow is $Q/2$. Finally letting $u_s = Q/2$ $(s = 1, 2, \ldots, S)$ gives a burst schedule for the original formulation in Eq. (3.1), where mobile devices always consume the data received in the, immediate, preceding subwindow. Therefore, this schedule satisfies the constraints in Eqs. (3.1c)–(3.1f), which yields the theorem. □

This theorem completes our proof of correctness. Next, we derive the approximation factor of the DBS algorithm and its time complexity.

Theorem 5 (Approximation Factor and Time Complexity)**.** *The DBS algorithm produces a near-optimal burst schedule with the number of bursts at most two times the number of bursts in the optimal schedule. Furthermore, the approximation factor of energy saving is given as*

$$\frac{\gamma^*}{\gamma} \leq \frac{1 - T_o R/SQ - 1/S}{1 - 2T_o R/SQ - 1/S}, \tag{3.6}$$

where γ^* and γ are the average energy saving achieved by the optimal scheduling algorithm and the DBS algorithm, respectively. Moreover, the DBS algorithm runs in time $O(pS \log(pS))$.

Proof. To compute the approximation factor, we first determine the number of bursts in the optimal schedule $\mathbf{L}^*$ and in the schedule $\mathbf{L}$ that is produced by the DBS algorithm. Consider any transformed formulation constructed by the for-loop in lines 2–4 with a set of subwindows $\mathbf{w}$. The number of subwindows is given as $|\mathbf{w}| = \sum_{s=1}^{S} \lceil 2pr_s/Q \rceil$. Notice that fewer preemptions in general leads to higher energy saving. However, to prevent buffer overflow instances, any optimal burst schedule $\mathbf{L}^*$ must have at least $\lceil pr_s/Q \rceil$ bursts for each video stream s, i.e., $n_s^* \geq \lceil pr_s/Q \rceil$. Thus, we have $|\mathbf{w}| \leq 2\,|\mathbf{L}^*|$. We then consider the number of bursts produced by the DBS algorithm. We note that the total number of bursts is bounded by the number of decision points, which are defined as the time instances at which either a subwindow starts or completes. Observe that, except for the boundary cases, a new subwindow is only *created* when the previous subwindow *completes*. This means that the total number of decision points is $|\mathbf{w}| + S \cong |\mathbf{w}|$. Since $|\mathbf{w}| \leq 2\,|\mathbf{L}^*|$, the number of bursts in $\mathbf{L}$ is at most 2 times the number of bursts in the optimal schedule $\mathbf{L}^*$. Thus, we have $\sum_{s=1}^{S} n_s \leq 2 \sum_{s=1}^{S} n_s^*$.

Now, we derive the approximation factor γ^*/γ as follows. Following the definition of the

average energy saving γ, we write the optimal energy saving as:

$$\gamma^* = 1 - \frac{T_o}{pS}\sum_{s=1}^{S} n_s^* - \frac{1}{RS}\sum_{s=1}^{S} r_s \approx 1 - \frac{T_o}{S}\frac{\sum_{s=1}^{S} r_s}{Q} - \frac{\sum_{s=1}^{S} r_s}{RS}. \tag{3.7}$$

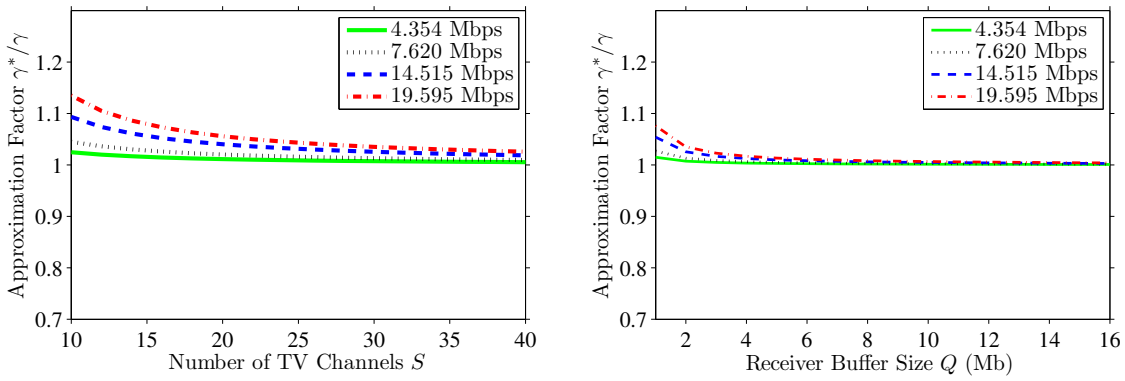
Similarly, we write the energy saving achieved by our algorithm as:

$$\begin{aligned}\gamma &= 1 - \frac{T_o}{pS}\sum_{s=1}^{S} n_s - \frac{1}{RS}\sum_{s=1}^{S} r_s \geq 1 - 2\frac{T_o}{pS}\sum_{s=1}^{S} n_s^* - \frac{1}{RS}\sum_{s=1}^{S} r_s \\ &\approx 1 - 2\frac{T_o}{S}\frac{\sum_{s=1}^{S} r_s}{Q} - \frac{\sum_{s=1}^{S} r_s}{RS}.\end{aligned} \tag{3.8}$$

While combining these two inequalities gives the approximation factor γ^*/γ, the resulting equation is too complex to reveal useful insights. To simplify it, we assume the bandwidth of the mobile video broadcast network is saturated: $R \approx \sum_{s=1}^{S} r_s$. This is a reasonable assumption, because the wireless spectrum is expensive, and service providers would broadcast as many video streams as possible. With this practical assumption, we get Eq. (3.6).

Last, we analyze the time complexity of the DBS algorithm. Observe that constructing the set $\mathbf{w}$ takes $O(|\mathbf{w}|)$, sorting subwindows on start times takes $O(|\mathbf{w}|\log|\mathbf{w}|)$, and storing outstanding subwindows in the priority queue and composing $\mathbf{L}$ take $O(|\mathbf{w}|\log|\mathbf{w}|)$. Moreover $|\mathbf{w}| = \sum_{s=1}^{S}(2pr_s/Q)$, and r_s/Q can be considered as a small constant for practical encoding bit rates (few hundreds kbps) and buffer sizes (few Mb). Thus, the time complexity of the DBS algorithm is $O(|\mathbf{w}|\log|\mathbf{w}|) = O(pS\log(pS))$. □

To shed some lights on the approximation factor derived in the above theorem, we numerically analyze it using a range of practical values. We consider mobile TV networks with various wireless medium capacities between 4.354 and 19.595 Mbps. These values cover possible bit rates of a 7 MHz frequency band with different modulation and coding schemes [16]. We let the overhead duration $T_o = 100$ msec. We first fix receiver buffer size at $Q = 2$ Mb and vary number of video streams between 10 to 40. We compute the approximation factor γ^*/γ and plot it in Figure 3.19(a) for different capacities of the wireless medium. The figure shows that our DBS algorithm produces very close results to the optimal ones. For example, using our algorithm to broadcast 10 to 40 video streams over a 7.620 Mbps medium, the average energy saving achieved by mobile devices is about 5% less than the absolute maximum energy saving that can be achieved using any algorithm to solve this NP-complete problem. Also, as detailed in the evaluation section, our algorithm



(a) Various number of TV channels

(b) Different receiver buffer size

Figure 3.19: The approximation factor of the DBS algorithm.

obtains these near-optimal results in the order of tens of milliseconds on a commodity PC. Notice also that, as the number of video streams increases, the approximation factor of our algorithm actually improves and approaches one. Next, we analyze the approximation factor as the receiver buffer Q varies from 1 to 16 Mb, while the number of video streams is fixed at 30. The results, shown in Figure 3.19(b), confirm that the approximation factor is typically close to 1, and it becomes even closer to 1 as the receiver buffer size increases, which is an expected trend in the future. These numerical results imply that our algorithm will yield almost optimal results in most deployments of mobile video broadcast networks.

We mention that γ^*/γ, is not *always* close to 1. More precisely, following Eq. (3.6), γ^*/γ becomes larger when $\frac{T_o}{SQ/R}$ increases. For example, γ^*/γ could be higher than 3 if $\frac{T_o}{SQ/R} \geq 0.4$. However, broadcast networks with $\frac{T_o}{SQ/R} \geq 0.4$ are *extreme cases*, as this inequality means that the overhead duration (T_o) of *one* burst is longer than 40% of the *total* burst length of all video streams. Energy saving in such networks would *already* be extremely low no matter how we construct burst schedules. Finally, we acknowledge that when $\frac{T_o}{SQ/R}$ is extremely small, even if there were more bursts scheduled, i.e., the number of scheduled bursts is more than two times the optimal number of bursts, the approximation factor of energy saving may not be much worse than that of the DBS algorithm. Nevertheless, DBS algorithm still outperforms all other algorithms, as we are not aware of any algorithm in the literature that leads to an approximation factor lower than the that of the DBS algorithm.

3.9 Evaluation of the Near-Optimal DBS Algorithm

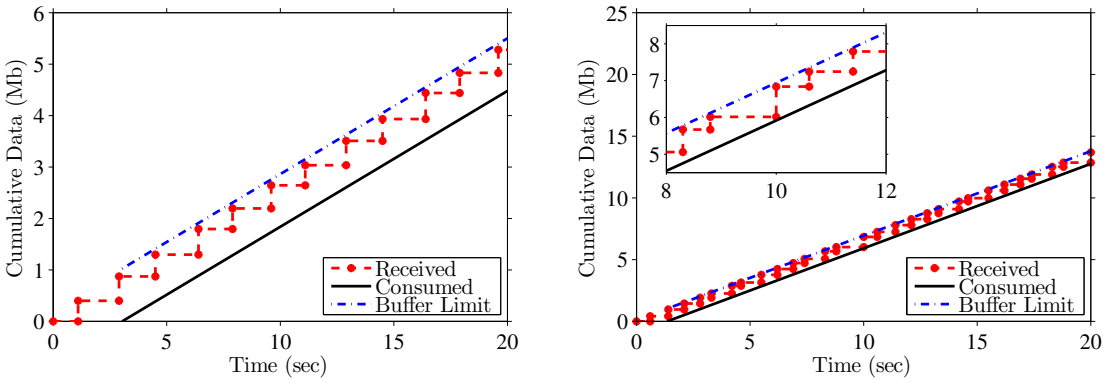
In this section, we conduct extensive experiments using a real mobile TV testbed as well as simulations to validate the correctness, efficiency, and near optimality of the proposed burst scheduling algorithm. We start by presenting the results from our testbed. Then, we use simulations to analyze wider ranges of several system parameters and their impact on the performance.

3.9.1 Setup of the Mobile TV Testbed

As illustrated in Section 3.6, we have set up a testbed for DVB-H networks for a realistic platform to analyze our burst scheduling algorithm. For the experiments, we configured the modulator to use a 5 MHz radio channel with QPSK modulation scheme. According to the DVB-H standard documents, this leads to 5.445 Mbps effective shared bandwidth [16]. We concurrently broadcast 12 TV channels using our algorithm for 10 minutes. We set the scheduling window length as 10 sec. The TV channel bit rates are randomly chosen between 200 and 800 kbps. The receiver buffer size is 1 Mb. For each TV channel, we set up a video streaming server on the base station to send 1-kB IP packets at the chosen bit rate. We set the overhead duration $T_o = 100$ msec. We collect detailed burst logs at the base station. The logs contain the start and end times (in msec) 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.

3.9.2 Experimental Results

Correctness of the DBS algorithm. We first validate the correctness of the proposed algorithm, i.e., we make sure that the algorithm results in no buffer violations for the receivers and no burst conflicts. For buffer violations, we compute the cumulative received bits (from the broadcasting base station) as the time progresses and compare this number against the cumulative consumed bits and the buffer upper limit. The cumulative consumed bits are computed by multiplying the bit rate of each TV channel with the time elapsed. The buffer upper limit is computed as the number of consumed bits plus the receiver's buffer size Q, which is set to 1 Mb. Sample results are presented in Figure 3.20 for two TV channels, results for other channels are similar. The figure shows the dynamics of the received bits, as the number of bits increases upon receiving a burst and then stays the same till the next



(a) Channel 1, 264 kbps

(b) Channel 4, 684 kbps

Figure 3.20: Buffer level dynamics of the resulting burst schedules of our algorithm: no over/underflow instances are observed.

burst. Meanwhile, the consumed bits and the buffer upper limits are continuously increasing with slope equals to the bit rate of the TV channel. The figure clearly shows that the curve representing the received bits never goes below the line representing consumed bits (i.e., no buffer underflow instances) and never exceeds the buffer upper limit (i.e., no buffer overflow instances). Note that these two figures show shorter time period, 20 sec, for the clarity of the figure. Nonetheless, this period covers multiple frames and the burst scheduling is identical in successive frames. Thus, the results are the same for the whole streaming period (10 minutes).

To check for burst conflicts, we compute time spacing between all bursts. We first sort bursts of all TV channels 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. We use the time spacing to validate that the resulting schedule leads to no burst conflicts, as a negative time spacing indicates bursts may intersect with each other. Our logs show that there are no buffer conflicts among bursts computed by our algorithm.

Energy saving and near-optimality of the DBS algorithm. We report the energy saving achieved by receivers of different TV channels when our burst scheduling algorithm is used. Figure 3.21(a) shows the energy saving of four representative TV channels; the energy saving of other channels are not shown for the clarity of the figure. We observe that the energy saving for low bit rate TV channels can be as high as 92%, while it is only 78% for high bit rate TV channels. This significant difference highlights the importance of

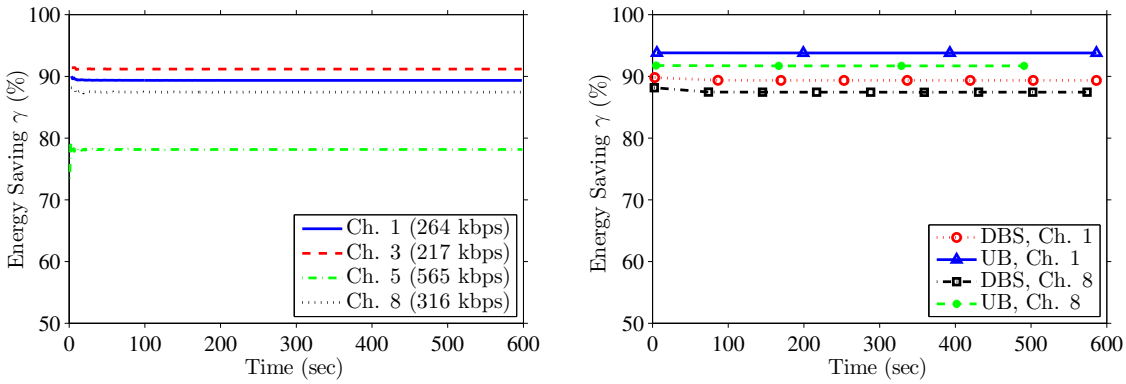


Figure 3.21: (a) Energy saving achieved by our algorithm for individual TV channels. (b) Comparing the energy saving achieved by our algorithm against a conservative upper bound on the energy saving.

choosing the appropriate bit rates to encode TV channels carrying diverse video content. The appropriate bit rate is not only important for enhancing the perceived visual quality, but it is also important for maximizing the energy saving and hence prolonging the viewing time on mobile devices.

Next, we compare the energy saving achieved by our algorithm against a very conservative upper bound on the maximum achievable energy saving. Recall that the burst scheduling problem is NP-complete and finding the exact optimal solution may take prohibitively long time to compute. We compute this upper bound as follows. For every TV channel, we make the base station broadcast only this channel without any other channels. The base station can maximize the energy saving by allocating the largest burst that can fill the receiver's buffer. The receiving circuit of the receiver is then turned off till the data of this burst is consumed. Clearly, this is a conservative upper bound on the energy saving that can be achieved by the receivers of the considered channel. This is because the base station has a complete freedom to allocate the largest burst without considering any interactions from other channels. We repeat this experiment 12 times; once for each considered TV channel. Then, we run our algorithm to compute the burst schedule for the 12 TV channels, and we make the base station broadcast all of them *concurrently*. We compute the energy saving achieved by mobile devices of each TV channel, and compare it against the upper bound on the energy saving. We report the results for two sample TV channels

in Figure 3.21(b); the results for other TV channels are similar. This figure shows that our algorithm produces near-optimal results: The gap between the energy saving achieved by our algorithm and the upper bound is less than 7% in all cases (including the ones not shown in the figure). We emphasize that this gap analysis is very conservative as we compare our algorithm, which concurrently broadcasts several TV channels at arbitrary bit rates, against the maximum energy saving of broadcasting a single TV channel.

Running time of the DBS algorithm. In all of the above experiments, our algorithm was running in real time on a commodity PC. The running time of our algorithm was in the order of tens of milliseconds. Thus, the algorithm can be invoked frequently as needed and in real time. This is a useful property for the network operators as it allows them to handle the dynamic nature of mobile TV networks and the usual changes in the offered TV programs. For example, broadcasting a commercial ad with high motion and rich visual content (and thus high bit rate) during a talk show (low bit rate) is quite simple: just before broadcasting the first burst of the commercial ad, our burst scheduling algorithm is invoked to compute a new burst transmission schedule considering the new bit rate of the ad. The same can be done for transitioning between shows and adding new TV channels.

Finally, we mention that the relative start time of each burst is recorded in the header of its predecessor burst such that the receivers know when they need to wake up to receive data [8, 10]. As the start time is sent in the relative form, its accuracy is not affected by any constant delays between the base station and its receivers. However, the start time is sensitive to the clock jitter caused by the inaccuracy of the timers of mobile devices. Therefore, mobile devices cope with this by waking up their receiving circuits slightly earlier to absorb the clock inaccuracy, which is referred to as *delay jitter*, and it is in the order of 10 msec [16].

3.9.3 Simulation Setup

We have implemented a simulator for mobile TV broadcast networks in Java. The simulator captures all important aspects relevant to the burst scheduling problem and it abstracts away details such as sending program guide to mobile devices, that are orthogonal to this problem. We developed the simulator to analyze wider ranges of the parameters, including extreme and boundary values that are difficult to exercise in the real testbed. This is useful to fully understand the merits and shortcomings of our algorithm.

Unless otherwise specified, we use the following parameters. The receiver buffer size

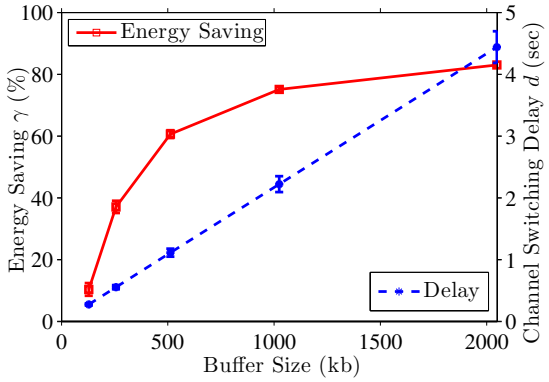


Figure 3.22: Impact of the receiver buffer on energy saving and switching delay.

$Q = 2$ Mb, wireless medium bit rate $R = 6.4$ Mbps, scheduling window length $p = 10$ sec, and overhead duration $T_o = 100$ msec. We randomly choose the bit rates of TV channels between 50 to 1,000 kbps to emulate different types of TV programs. We repeat each experiment 100 times, and we report the means and 95% confidence intervals of the performance metrics.

We consider several performance metrics, including: energy saving γ, channel switching delay d, and running time of our algorithm. The channel switching time is an important metric in mobile TV networks, as many users tend to flip through several channels before they decide on a specific channel to watch. We define the channel switching delay as the time a user waits before s/he starts viewing a selected channel when a change of channel is requested by that user. The channel switching delay is composed of several accumulated parts, in which the frame refresh delay and time slicing delay are the two dominating contributors [29, 33]. The frame refresh delay refers to the time period between receiving the first bit of a new video stream and receiving the next random access point, typically an intra-coded frame, of that video. The time slicing delay refers to the time period between locking on a mobile TV signal and receiving enough bursts of the selected TV channel for a smooth playout. Our simulator captures only the time slicing delay. The frame refresh delay is difficult to simulate, because it depends on the specific video content, how it is encoded, and how the frames are organized. In addition, The time slicing delay is a direct outcome of our burst scheduling algorithm, while the frame refresh delay is orthogonal to our algorithm.

3.9.4 Simulation Results

Tradeoff between energy saving and channel switching delay. Our results reveal a trade off between the achieved energy saving by mobile devices and the average channel switching delay. Furthermore, this tradeoff can be controlled by choosing the appropriate receiver buffer size. To demonstrate this tradeoff, we vary the receiver buffer size between 128 kb and 2,048 kb. We run our simulator and compute the energy saving and the channel switching delay for each buffer value. The results of channel switching delay are shown in Figure 3.22. This figure shows that smaller channel switching delays—which are desirable for better viewing experience—require smaller receiver buffer sizes. For example, the average switching delay is reduced from 4 to 2 sec by reducing receiver buffer size from 2 to 1 Mb. This indicates that changing the buffer size can control channel switching delays. However, smaller receiver buffer sizes dictate shorter bursts, which increases the energy consumption as the receiving circuit of the receivers needs to wake up more often for smaller bursts and in each time it incurs an additional overhead (of at least T_o msec). This is also shown in Figure 3.22 as the energy saving diminishes when receiver buffer size becomes very small. The figure indicates that a buffer size of at least 1,000 kb is needed to achieve an average energy saving of 75%. With 1,000 kb buffer, the average switching delay is about 2 sec. If further smaller switching delays are desired without sacrificing the energy saving, other mechanisms may be needed. For example, scalable video coding can be used to encode each TV channel into multiple layers [34]. This scalable video coding, however, imposes an additional overhead as it consumes a small fraction of the wireless medium bandwidth.

Impact of wireless medium utilization. Next, we evaluate the performance of our algorithm under different bandwidth utilizations of the shared air medium. We consider various bandwidth utilization: from 30% to 100% to cover all practical scenarios. The scheduling window length is fixed at 10 sec. For each bandwidth utilization, we construct burst scheduling problems with TV channels encoded at arbitrary bit rates randomly chosen between 50 and 1,000 kbps. We then solve each burst scheduling problem using our DBS algorithm and measure the energy saving and the switching delay. The results , shown in Figure 3.23, imply that increasing the bandwidth utilization has a minor impact on the energy saving and the switching delay. For example, the average energy saving (figure not shown here due to space limitations) is reduced by less than 5% as the utilization increases from 30% to 100%, while Figure 3.23(a) shows that the channel switching delay decreases

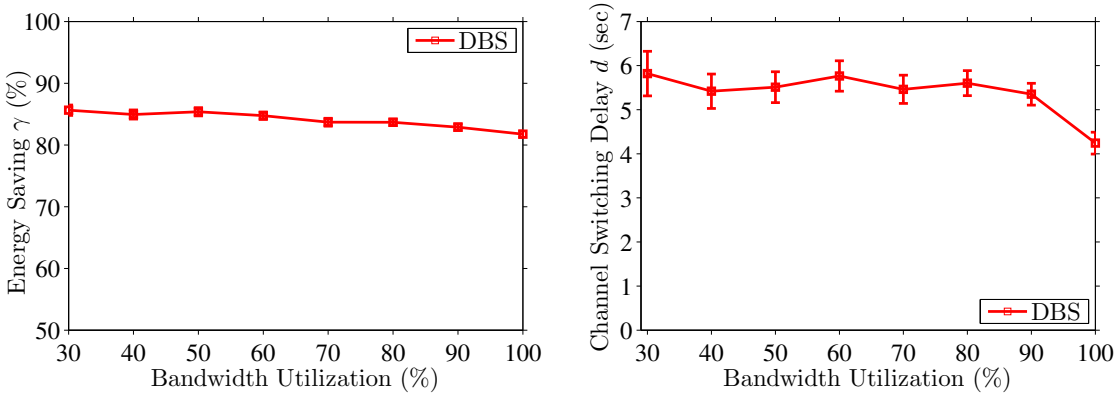


Figure 3.23: The implication of bandwidth utilization on: (a) energy saving, and (b) channel switching delay.

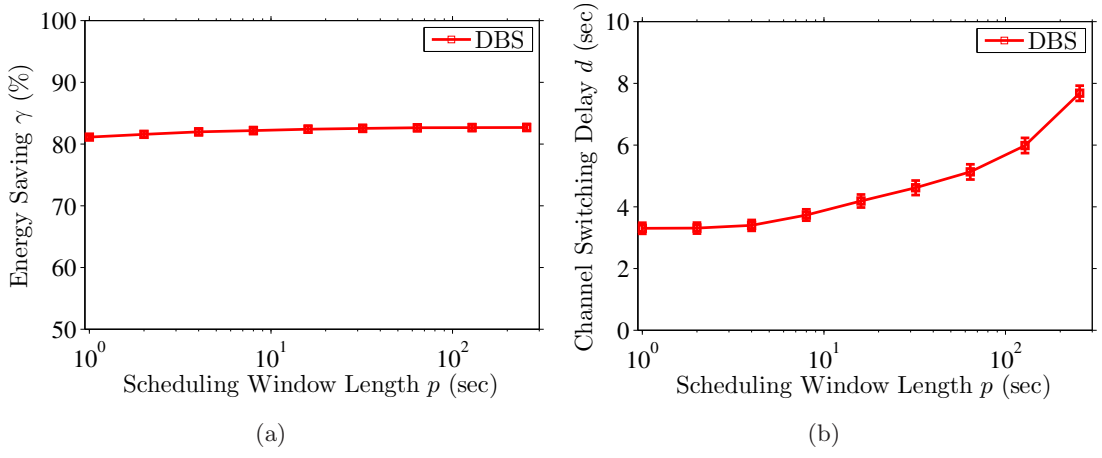
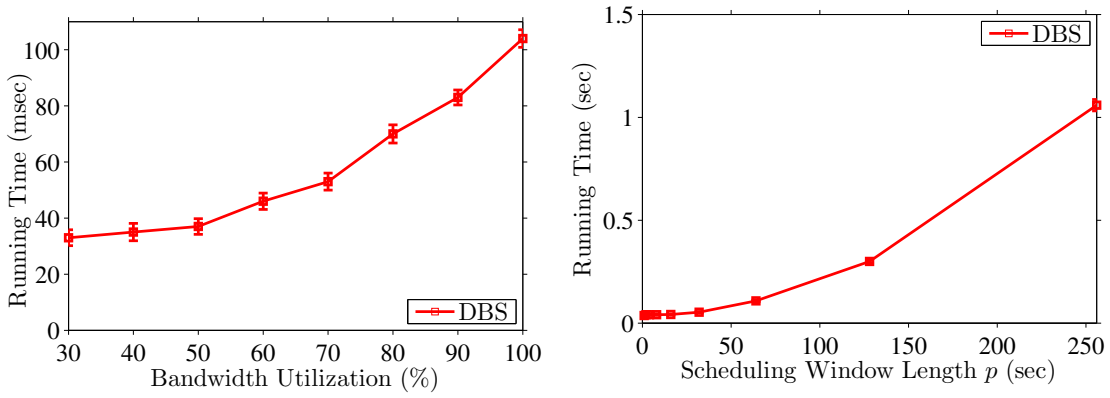


Figure 3.24: The implication of scheduling window length on: (a) energy saving, and (b) channel switching delay.



(a) (b)

Figure 3.25: Running time of our algorithm under different: (a) medium utilization values, and (b) scheduling window lengths.

from 5.8 to 4.2 sec as the utilization increases. This degradation of energy saving is intuitive, because heavy loaded mobile TV networks leave smaller rooms for arranging the bursts of different TV channels to save energy, which results in more bursts for each TV channel. More bursts, however, lead to lower channel switching delay because mobile devices can reach the next burst faster. Most importantly, this set of experiments shows that our algorithm is robust and functions properly even in fully loaded networks.

Impact of scheduling window length. We analyze the impact of various scheduling window lengths on the performance. We vary the scheduling window length from 10 sec to 4 min, and measure the energy saving and the channel switching delay in each case. We fix the bandwidth utilization at 90%. We report the results in Figure 3.24, which shows that increasing the scheduling window length from 10 sec to 4 min improves the average energy saving by only about 2%, while it doubles the average channel switching delay. The marginal energy saving improvement is clearly not desirable given the significant increase in the switching delay. This experiment indicates a scheduling window length in the range between 10 and 60 sec would achieve a high energy saving without incurring excessive channel switching delays. The specific value of the scheduling window can be decided by the network operator based on other considerations, such as the desired level of responsiveness to changes in the video content of the TV channels.

Running time. Finally, we report the average running time of our algorithm on a commodity PC with a 2.6 GHz processor that runs Linux. Figure 3.25(a) shows the

running time of the algorithm to compute the burst transmission schedules as the bandwidth utilization varies from 30% to 100%, while the scheduling window length is fixed at 10 sec. The figure shows that the maximum running time is less than 110 msec on a *commodity* PC, to compute burst schedules for fully-utilized mobile TV network. These results confirm our results from the real testbed that our algorithm can indeed run in real time.

We also collect the running time of our algorithm as the scheduling window length varies from 10 to 256 sec, while the bandwidth utilization is fixed at 90%. We plot the results in Figure 3.25(b). This figure shows that for practical scheduling window lengths (less than 60 sec), our algorithm terminates in the order of milliseconds. Larger frames contain many bursts and require more computations to carefully specify the start and end of each of them. But even for such (unusual) large frames, our algorithm terminates in less than 1.1 sec. Notice also that the running time of 1.1 sec is insignificant compared to a scheduling window length of 256 sec, because our algorithm is invoked at most once in each scheduling window.

3.10 Conclusions

In this chapter, we studied the energy optimization problem in mobile video broadcast networks, in which a base station broadcasts multiple video streams in bursts with bit rates much higher than the encoding rates of the video streams. We showed that the energy optimization problem is NP-complete when video streams have arbitrary bit rates. We then solved the problem in two steps. We first studied a practical simplification of the problem, which enables the use of different classes for video streams, where each class has a different bit rate. The bit rate of class c, r_c, can take any value in the form of $r_c = 2^i \times r_1$, where $i \in \{0, 1, 2, 3, \dots\}$, and r_1 is the bit rate of the lowest class. r_1 can take any arbitrary bit rate. We showed that this classification can result in better viewing experience by reducing quality variation among all video streams. We did this by encoding various video sequences with diverse content complexities and empirically analyzing their R-D characteristics. We proposed an optimal burst scheduling algorithm, called P2OPT (Power-of-Two Optimal), for the simplified problem, which runs in $O(S \log S)$ time, where S is the number of video streams. We proved the correctness and optimality of the P2OPT algorithm. We derived closed form equations for the energy saving achieved by the P2OPT algorithm and the resulting channel switching delay. We numerically analyzed theses equations to demonstrate

the existence of a tradeoff between energy saving and channel switching delay, and to show that this tradeoff can be controlled by the receiver's buffer size. In addition, we evaluated the performance of the P2OPT algorithm using simulations, and we showed that it achieves an average energy saving of more than 92%. We implemented the P2OPT algorithm in an actual mobile TV testbed. We analyzed several logs from the testbed and empirically showed that our P2OPT algorithm can run in real-time, results in no buffer under/overflow instances and no burst conflicts, and yields optimal energy saving.

Next, we studied the general form formulation of the energy optimization problem in mobile video broadcast networks, in which multiple video streams with arbitrary bit rates are concurrently broadcast by a base station. We proposed a novel approximation algorithm, called DBS (Double Buffering Scheduling), for this burst scheduling problem. We proved that the DBS algorithm has a small approximation factor, and it has a time complexity of $O(pS \log(pS))$, where S is the number of video streams , and p is the scheduling window length on which the burst scheduling problem is solved. We implemented and validated our DBS algorithm in a real mobile TV testbed that complies to the DVB-H standard. We also developed a simulator for mobile video broadcast networks to study the impact of wide ranges of various parameters on the performance of the DBS algorithm. Our experimental and simulation results show that the resulting schedules are correct and the approximation factor of the DBS algorithm is very close to one for most practical mobile video broadcast networks. They also verify that the DBS algorithm can run in real time and it scales well to large scheduling problems.

Chapter 4

Goodput Optimization

In this chapter, we solve the goodput optimization problem in both open- and close-loop networks, where a base station concurrently broadcasts multiple VBR streams to mobile devices. We formulate this problem into a multiobjective optimization problem and show its hardness. We propose two algorithms to solve this problem, which are suitable for different broadcast networks. We analyze the proposed algorithms, and we evaluate them using simulations and a real mobile TV testbed.

4.1 Introduction

We study the goodput maximization problem stated in Problem 2. This problem is to construct burst schedules for the multiplexer in a broadcast network that transmits VBR streams in order to achieve the following three goals:

1. *Streaming Quality:* Video streaming is a real-time application. Video data that miss their decoding deadlines cannot be rendered to users, and are essentially useless. In addition, video data that arrive too early may use up the buffer space, and prevent newly received data from being buffered. Therefore, sending video data too early or too late both lead to playout glitches and may drive users away from the service. Hence, multiplexers should ensure that there is no buffer violation instance 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:* Mobile devices are battery-powered, and thus are vulnerable to high energy consumption, which leads to shorter watch time before users replacing or recharging their batteries. Shorter watch time leads to more toxic waste of primary (non-rechargeable) or rechargeable batteries, and thus is not environmentally friendly. Hence, multiplexers should reduce the energy consumption on mobile devices to increase user satisfaction and reduce pollution.

3. *Goodput:* The wireless spectrum is expensive: licensing a spectrum usually costs many millions of dollars annually. Therefore, 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 ontime over the network capacity. Higher goodput in general leads to more video streams concurrently broadcast within a given spectrum. Hence, multiplexers should achieve high goodput to increase the net profits of network operators.

We solve Problem 2 in this chapter. We first formulate this problem 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. 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. Although the proposed algorithm minimizes the playout glitches in open-loop broadcast networks, it may result in some playout glitches when the total bit rate of all video streams exceeds the network bandwidth. To address this issue, we propose a new scheduling algorithm for open-loop networks. The new scheduling algorithm employs longer lookahead windows and utilizes the air medium time to reduce the number of playout glitches. We use trace-drive simulations to show the effectiveness of this new scheduling algorithm.

4.2 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 [19] and [16] study the energy saving achieved by a given burst schedule. These two works do not solve the burst scheduling problem. In Chapter 3, 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 also present an efficient burst scheduling algorithm for video streams that can take any arbitrary bit rates. The burst scheduling algorithms proposed in Chapter 3 target CBR video streams, and do not consider the rate variability within each video stream. In this chapter, we solve the burst scheduling problem for VBR video streams, which may lead to better streaming quality, shorter delay, and more concurrent broadcast streams [27]. More importantly, in this chapter, we also maximize the goodput in broadcast networks, in addition to maximize energy saving for mobile devices.

Streaming VBR videos in the Internet is challenging. Several smoothing algorithms have been proposed in the literature, e.g., in [100, 101], 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 [50,62], 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 chapter. Camarda et al. [102] 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 [50, 62, 100–102] 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 [29, 57–59, 103], propose joint coder designs for popular video coding standards. Wang and Vincent [103] propose a joint coder for MPEG-2 coded streams. Tagliasacchi et al. [57] and He and Wu [58] propose joint coders for H.264/AVC coders. Rezaei et al. [29]

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. [59] 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 chapter, we solve the burst scheduling problem in broadcast networks with and without joint coders.

Rezaei et al. [104] 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, i.e., without the constraints of scheduling windows. This gives us opportunities to achieve better energy saving. Second, we assume that there is a small lookahead 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 lookahead window is a reasonable assumption because many programs are pre-recorded, while live streams are often delayed to allow censoring and editing. Since the work in [104] probabilistically schedules bursts, it has to wake up receiving circuits earlier to accommodate any prediction errors. Therefore, 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 actual frame sizes which are more accurate than the predicted frame sizes. Therefore, we only compare our algorithm against the current base stations.

4.3 Problem Formulation

4.3.1 Problem Statement and Hardness

We list all symbols used in the chapter in Table 4.1 for quick reference. We study broadcast networks in which a base station transmits S video streams to many mobile devices over a

Table 4.1: List of symbols used in goodput optimization.

Sym	Definition	Sym	Definition
T_o	overhead duration	f_k^s	time of burst k for s
S	number of video streams	b_k^s	size of burst k for s
R	burst bit rate	c_k^s	buffer level at time f_k^s
T	broadcast time	$\mathbf{L}$	burst schedule
I	no. frames	n_s	no. bursts for s
F	frame rate	m_p^s	last frame for window p of s
Q	receiver buffer size	y_p^s	no. bits for window p of s
l_i^s	size of frame i of stream s	x_p^s	start time for window p of s
γ	energy saving	z_p^s	end time for window p of s
σ	goodput	d	initial delay

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, i.e., $l_i^s F < R$. 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 [16,19]. 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 can re-state Problem 2 as: find the optimal burst schedule for S concurrent VBR video streams to maximize the goodput σ *and* the energy saving γ, without resulting in any playout glitches on mobile devices.

In this optimization 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 2 is a generalization of Problem 1, 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 in Theorem 1. Therefore, Problem 2, 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 tradeoff among objective functions [105, 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 4.4, 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.

4.3.2 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, e.g., in mobile TV broadcast networks, T_o ranges from 50 to 250 msec [9, 10, 16]. Moreover, in Section 2.6, we empirically show that a recent Nokia cellular phone has T_o in the range of 80–140 msec. 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 $h/F \le f_k^s$. 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:

$$Pri: \max_{\mathbf{L}} \quad \sigma = \frac{\sum_{s=1}^{S}\sum_{j=1}^{n_s} b_j^s \Big/ R}{I/F}; \tag{4.1a}$$

$$Sec: \max_{\mathbf{L}} \quad \gamma = 1 - \frac{\sum_{s=1}^{S}\sum_{k=1}^{n_s}(T_o + b_k^s/R)}{I/F}\Big/ S; \tag{4.1b}$$

$$\text{s.t.} \quad \left[f_k^s, f_k^s + \tfrac{b_k^s}{R}\right) \bigcap \left[f_{\bar{k}}^{\bar{s}}, f_{\bar{k}}^{\bar{s}} + \tfrac{b_{\bar{k}}^{\bar{s}}}{R}\right) = \varnothing; \tag{4.1c}$$

$$c_k^s > 0; \tag{4.1d}$$

$$c_k^s + b_k^s - \textstyle\sum_{f_k^s \le j/F < f_k^s + b_k^s/R} l_j^s \le Q; \tag{4.1e}$$

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

In this formulation, the primary goal is to maximize the 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. (4.1c)–(4.1e) guarantee that the resulting burst schedule is feasible. In particular, Eq. (4.1c) ensures that there are no burst intersections among all S video streams. Eq. (4.1d) checks the buffer level for stream s at the start time of every burst to prevent buffer underflow instances. Eq. (4.1e) 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.

4.4 Problem Solution

We propose, in Section 4.4.1, an approximation algorithm to solve the burst scheduling problem. In Section 4.4.2, we show that our algorithm achieves optimality along one objective function (goodput) and near-optimality along the other objective function (energy saving). In Section 4.4.3, we describe some practical considerations when implementing the proposed algorithm in actual base stations.

4.4.1 Scheduling Algorithm for VBR Streams

The high-level idea of our algorithm is similar to the one described in Section 3.8. 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

$2 \leq p \leq p_s$, 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 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 ($1 \leq p \leq p_s$), 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:

$$\begin{cases} m_p^s = 0, & p = 0; \\ \sum\limits_{j=m_{p-1}^s+1}^{m_p^s} l_j^s \leq \frac{Q}{2} < \sum\limits_{j=m_{p-1}^s+1}^{m_p^s+1} l_j^s \,, & \forall\, 1 \leq p \leq p_s. \end{cases} \tag{4.2}$$

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 $1 \leq s \leq S$ and $1 \leq p \leq p_s$. Letting y_p^s be the aggregate data amount that must be received in window p, we can write y_p^s as:

$$y_p^s = \sum_{j=m_{p-1}^s+1}^{m_p^s} l_j^s. \tag{4.3}$$

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:

$$x_p^s = \begin{cases} 0, & p = 1; \\ (m_{p-2}^s + 1)/F, & 2 \leq p \leq p_s, \end{cases} \tag{4.4}$$

$$z_p^s = \begin{cases} \sum_{s=1}^{S} y_1^s / R & p = 1; \\ m_{p-1}^s / F, & 2 \leq p \leq p_s. \end{cases} \tag{4.5}$$

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 $\sum_{s=1}^{S} y_1^s/R$, which is the shortest possible window length to send data in the first window of all video streams. Since $y_1^s \leq Q/2$ (indicated by Eqs. (4.2) and (4.3)), the delay incurred by the SMS algorithm is bounded by

$$d = (SQ)/(2R). \tag{4.6}$$

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

$$Pri: \max_{\mathbf{L}} \quad \sum_{s=1}^{S} \sum_{j=1}^{n_s} b_j^s; \tag{4.7a}$$

$$Sec: \min_{\mathbf{L}} \quad \sum_{s=1}^{S} n_s; \tag{4.7b}$$

$$\text{s.t.} \quad y_p^s = \sum_{\forall\, x_p^s \leq f_k^s < z_p^s} b_k^s; \tag{4.7c}$$

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

This formulation first maximizes the goodput by maximizing the amount of ontime delivered video data in Eq. (4.7a). It then maximizes the energy saving by minimizing the number of bursts in Eq. (4.7b), as each burst incurs a constant overhead duration T_o. The constraint in Eq. (4.7c) 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, 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 $z^s_{p'} < t$ 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^s_p.

We call this algorithm Statistical Multiplexing Scheduling (SMS) algorithm, and give its high-level pseudocode in Figure 4.1. 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 lookahead window (in the order of a few seconds) for frame size l^s_i, 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 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.

4.4.2 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 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.

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. (4.2)—(4.5)
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
8.    among all windows p' with x_{p'}^s ≤ t and z_{p'}^s > t, and t is the current time, t_n is the
8.    time of the 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. (4.2)—(4.5)
11. }
```

Figure 4.1: An efficient burst scheduling algorithm.

Theorem 6 (Correctness)**.** *The SMS algorithm returns a feasible burst schedule for the original burst scheduling problem (Problem 2) in closed-loop broadcast networks.*

Proof. 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. (4.7c) 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, i.e., any burst scheduling algorithm that achieves optimal goodput leads to no late data. Next, we borrow the result from Theorem 7, which states the SMS algorithm maximizes the goodput. This means that the SMS algorithm produces burst schedules with late data, i.e., $y_p^s \leq \sum_{\forall\, x_p^s \leq f_k^s < z_p^s} b_k^s$, which yields Eq. (4.7c). 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. (4.2)), the resulting schedule leads to no buffer violation instances in the original problem. □

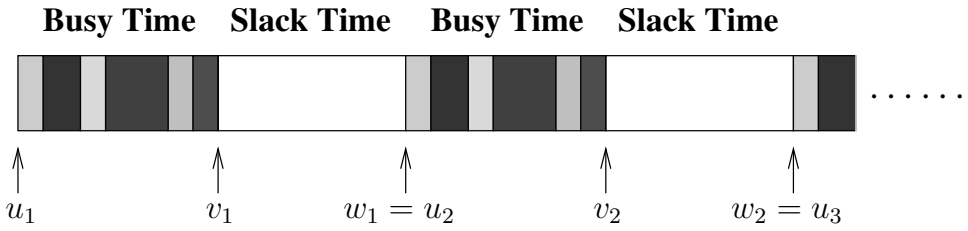


Figure 4.2: The resulting schedule of the SMS algorithm consists of interleaved busy and slack time periods. Different shaded blocks represent bursts for different video streams.

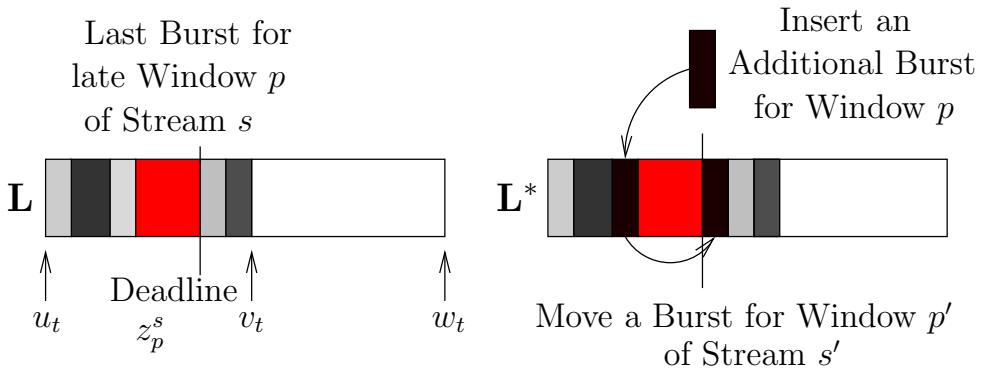


Figure 4.3: Inserting a burst requires moving another burst, as there is no gap between bursts in busy time periods.

Theorem 7 (Optimality of Goodput)**.** *The SMS algorithm produces optimal burst schedules in terms of goodput.*

Proof. 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 4.2. 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 ontime. 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 4.3. 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* θ-sec burst to window p, where $\theta > 0$. By definition, goodput only considers video data that arrive ontime, 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 $z_{p'}^{s'} \leq z_p^s$. 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 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. □

Theorem 8 (Near-Optimality of Energy Saving)**.** *The SMS algorithm produces a near-optimal burst schedule with the number of bursts at most two times the number of bursts in the optimal schedule. Moreover, the approximation gap of energy saving is written as:*

$$\Delta\gamma = \gamma^* - \gamma \leq T_o r/Q, \tag{4.8}$$

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.

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 $n_s^* \geq \sum_{i=1}^{I} l_i^s/Q$. Then, following the definition of energy saving, we write the energy saving of stream s as:

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

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:

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

Next, we let n_s be the number of bursts scheduled for s by the SMS algorithm. Based on Eq. (4.2), we use $\delta_p^s = \frac{Q}{2} - \sum_{j=m_{p-1}^s+1}^{m_p^s} l_j^s$ 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 $p_s = \sum_{i=1}^{I} l_i^s \Big/ (Q/2)$. 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 $\sum_{s=1}^{S} n_s \leq \sum_{s=1}^{S} p_s$. 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 $\sum_{s=1}^{S} n_s \leq \sum_{s=1}^{S} p_s = 2 \sum_{s=1}^{s} \sum_{i=1}^{I} l_i^s / Q$, we have: $\gamma \geq 1 - (\frac{2T_o}{Q} + \frac{1}{R})r$. Combining γ and γ^* yields the theorem. □

Theorem 9 (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.*

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 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 $O(PS + S^2) + O(PS) = O(PS + S^2)$. □

The above 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 8 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

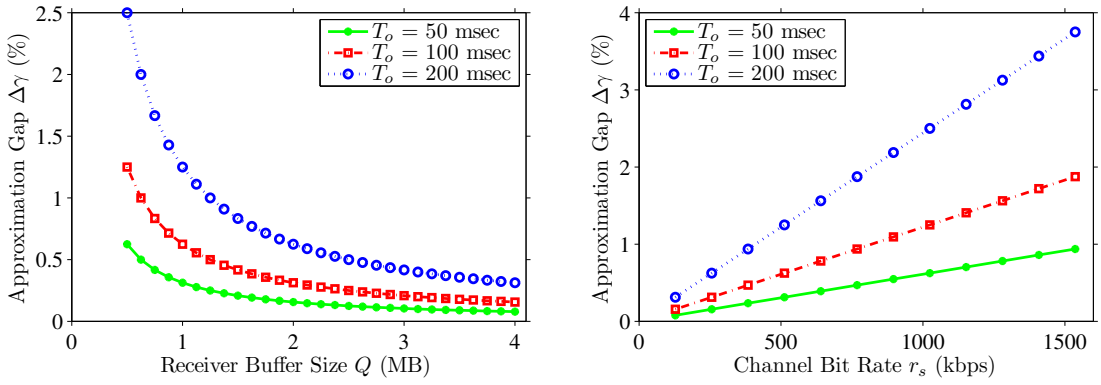


Figure 4.4: The proposed algorithm leads to small approximation gap with typical parameters: (a) average coding bit rate is 512 kbps, and (b) receiver buffer is 1 MB.

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 Figure 4.4. Figure 4.4(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. Figure 4.4(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.

We mention that $\Delta\gamma$ is not *always* small. More specifically, following Eq. (4.8), $\Delta\gamma$ becomes larger when $\frac{T_o}{Q/r}$ increases. However, broadcast networks with large $\frac{T_o}{Q/r}$ are *extreme cases*, and in these networks the ratio of overhead duration (T_o) and burst length (Q/r) is large. Therefore, no matter how we construct burst schedules, energy saving in such networks would be very low. In addition, we acknowledge that when $\frac{T_o}{Q/r}$ is extremely small, even if there were more bursts scheduled, i.e., the number of scheduled bursts is more than

two times the optimal number of bursts, the approximation gap of energy saving may not be much worse than that of the SMS algorithm. Nevertheless, SMS algorithm still outperforms all other algorithms, as we are not aware of any algorithm in the literature that leads to an approximation gap lower than the that of the SMS algorithm.

Last, we comment on the delay incurred by the SMS algorithm, which is bounded by $(SQ)/(2R)$ as shown in Eq. (4.6). 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 512 kbps. 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. We emphasize that the SMS algorithm does not employ smoothing buffers to regulate the bit rates of VBR streams. This is different from the RVBR algorithm described in Section 2.5 and used in current base stations. Hence, the SMS algorithm does *not* suffer long preroll delay due to the smoothing buffer, which can be more than 400 sec as illustrated in Figure 2.9(b).

4.4.3 Practical Considerations

Wireless networks may impose limitations on the burst size. For example, DVB-H standard specifies that a burst cannot exceed the maximum burst size of 2 Mb [9]. Broadcast networks may also have constraints on minimum burst size, because short bursts require higher receiver sensitivity, and could lead to higher transmission error rates [73]. Furthermore, as we mentioned earlier, shorter bursts in general result in lower energy saving, thus should be avoided. Incorporating maximum and/or minimum burst size in the SMS algorithm can be done by slightly changing line 7 of Figure 4.1. First, instead of next decision point, a burst may terminate early to comply to the maximum burst size. Second, any decision points that would lead to a burst shorter than the minimum burst size are suppressed for that iteration. Therefore, the SMS algorithm is general and can support constraints on burst size, which are network-specific parameters.

4.5 Scheduling in Open-Loop Broadcast Networks

In this section, we consider the problem of broadcasting VBR streams in open-loop broadcast networks , in which the aggregate bit rate of all video streams may occasionally exceed the

network bandwidth. In Section 4.5.1, we first show that the proposed SMS algorithm can be used in open-loop broadcast networks, and give the sufficient condition of the resulting burst schedules to be feasible, i.e., leads to no playout glitches. For general open-loop broadcast networks, we propose in Section 4.5.2 another burst scheduling algorithm that employs a larger lookahead window to mitigate the potential late frames due to overloading the broadcast network.

4.5.1 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 4.1 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 gives glitch-free burst schedules, i.e., 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 6, 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 7.

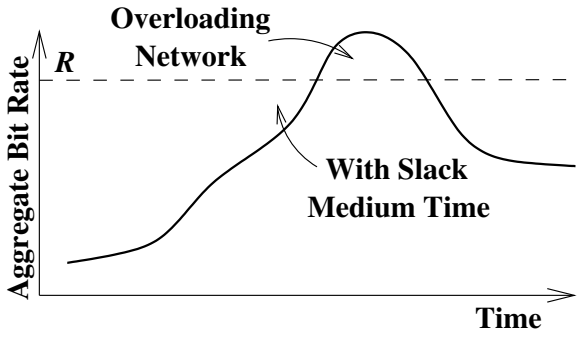


Figure 4.5: Video streams in open-loop networks may overload the broadcast network, which may in turn lead to buffer underflow instances on mobile devices.

4.5.2 Algorithm for Glitch-Free Playouts

The condition in Corollary 1 may not hold in heavily-loaded open-loop broadcast networks. Therefore, the SMS algorithm could drop some video data when the broadcast network is overloaded. Specifically, when the aggregate bit rate of all video streams is higher than the network bandwidth, the SMS algorithm may suffer from late windows in line 9 of Figure 4.1. The SMS algorithm will move on to the next window, because the late video data can no longer be rendered to users. This in turn lead to buffer underflow instances and playout glitches. Figure 4.5 shows an illustrative example of the aggregate bit rate of an open-loop network. There is a period of time the aggregate bit rate exceeds the broadcast network bandwidth, and video streams are vulnerable to playout glitches during this time period. Therefore, a better burst scheduling algorithm is needed for open-loop broadcast networks.

We make an observation on the illustrative bit rate curve in Figure 4.5. We notice that the aggregate bit rate only occasionally overloads the broadcast network, while the broadcast network often has slack medium time. Therefore, we propose to utilize the slack medium time, *before* the broadcast network is overloaded, to transmit the video data that will be otherwise dropped by the multiplexer due to overloaded network. Sending some video data using slack medium time enables us to absorb the aggregate bit rate spikes that exceed the broadcast network bandwidth.

We propose an SMS′ burst scheduling algorithm to solve the scheduling problem in open-loop broadcast networks. While the SMS′ algorithm is based on the SMS algorithm, they have two major differences. First, the SMS′ algorithm keeps track of and utilize slack medium time to mitigate playout glitches caused by overloaded broadcast networks.

Second, the SMS′ employs a longer lookahead window to construct burst schedules. The SMS′ algorithm needs a longer lookahead window for early detection of aggregate bit rate spikes. The length of this lookahead window ζ is a system parameter. Longer lookahead window gives the SMS′ algorithm more opportunities to mitigate buffer violation instances, but may not be suitable for live programs.

The SMS′ algorithm works as follows. We first construct the transformed formulation, and then iterate through all decision points (defined in Section 4.4.1). At each decision point, we schedule a burst to the outstanding window with the smallest end time. The burst length is from the current decision point to the next decision point. If there is no outstanding window with a start time earlier than the current decision point, there is no video data to schedule and thus we have a slack time slot. We push this slack time slot into a stack for later usage, and move to the next decision point.

Upon reaching a decision point which is also the end time of a window p_s we check whether p_s is complete, i.e., whether we have delivered all p_s's data ontime. If window p_s has any residue data to send, we pop a slack time slot from the stack and allocate a burst for video stream s. Since slack time slots are on a stack, we always get the most recent slack time slot, which reduces the chance for buffer overflow instances on mobile devices receiving stream s. Depending on the amount of residue data in window p_s, it may require one or more slack time slots. We iteratively pop slack time slots for p_s until it is complete. We mention that when allocating a slack time to a video stream, we need to validate whether the new burst would lead to buffer overflow instances on mobile devices. This can be done by computing the buffer level at the end of the new burst using Eq. (4.1e). Slack time slots that have been fully used are discarded. Partially used slack time slots are pushed back to the stack after adjusting its start and end times. Slack time slots in the past are also discarded as they can no longer be used. We move from the end time of window p_s to the next decision point when: (i) the p_s is complete or (ii) no more bursts can be allocated.

We present the high-level pseudocode of the SMS′ algorithm in Figure 4.6. This algorithm constructs the first window for each video stream in lines 3–5. It uses the for-loop between lines 7 and 16 to iterate through all decision points. In line 8, it checks whether there is any outstanding window. If not, it inserts a slack time slot into the stack in line 9; otherwise, it schedules a burst to video stream s in line 11. The if statement in lines 12 and 13 allocates slack time slots to any late window, until that window is complete or there is no more bursts can be allocated. New window is generated in line 18 upon a window is

The 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. (4.2)—(4.5)
6. // burst scheduling
7. **foreach** decision point of window p for stream s
8. **if** there is no window p' with $x_{p'}^s \leq t$ and $z_{p'}^s > t$, where t is the current time
9. push a slack time slot into the stack
10. **else**
11. schedule a burst from times t to t_n for s, where window p of s has the smallest z_p^s
11. among all windows p' with $x_{p'}^s \leq t$ and $z_{p'}^s > t$, and t_n is the time of the next
11. decision point
12. **if** window p of s is late
13. allocate one or more bursts to stream s using the slack time slots in the stack
14. **if** window p of s completes or is late
15. generate a new window p for s and determine x_p^s, y_p^s, and z_p^s using Eqs. (4.2)—(4.5)
16. }

Figure 4.6: A burst scheduling algorithm for open-loop broadcast networks.

complete or late. The algorithm terminates once there is no other decision points.

We comment on the time complexity of the SMS$'$ algorithm. Compared to the SMS algorithm, the only additional complexity of the SMS$'$ algorithm is the slack time slots operation. We note that the total number of slack time slots is bound by $O(PS)$, where P is the maximum number of windows among all S video streams. We consider the size of the stack is a small constant. This is because the stack size is limited by the lookahead window, which is usually not too long. However, if the lookahead window is long, such as for pre-recorded videos, we can also enforce a maximum stack size in order to bound the complexity. Since stack operations are constant, we know the additional time complexity imposed by managing slack medium time is $O(PS)$. Combining this with Theorem 9, we know the SMS$'$ algorithms also has a small time complexity of $O(PS + S^2)$.

4.6 Simulation

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

4.6.1 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 and SMS$'$ algorithms in the simulator. We have also implemented the current VBR$_\alpha$, RVBR$_\beta$, and DVBR$_\tau$ algorithms (which are described in Section 2.5) 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 5 MHz channel bandwidth. This gives us a broadcast network with bandwidth $R = 17.2$ Mbps [16]. We consider an overhead duration $T_o = 100$ msec and receiver buffer size $Q = 4$

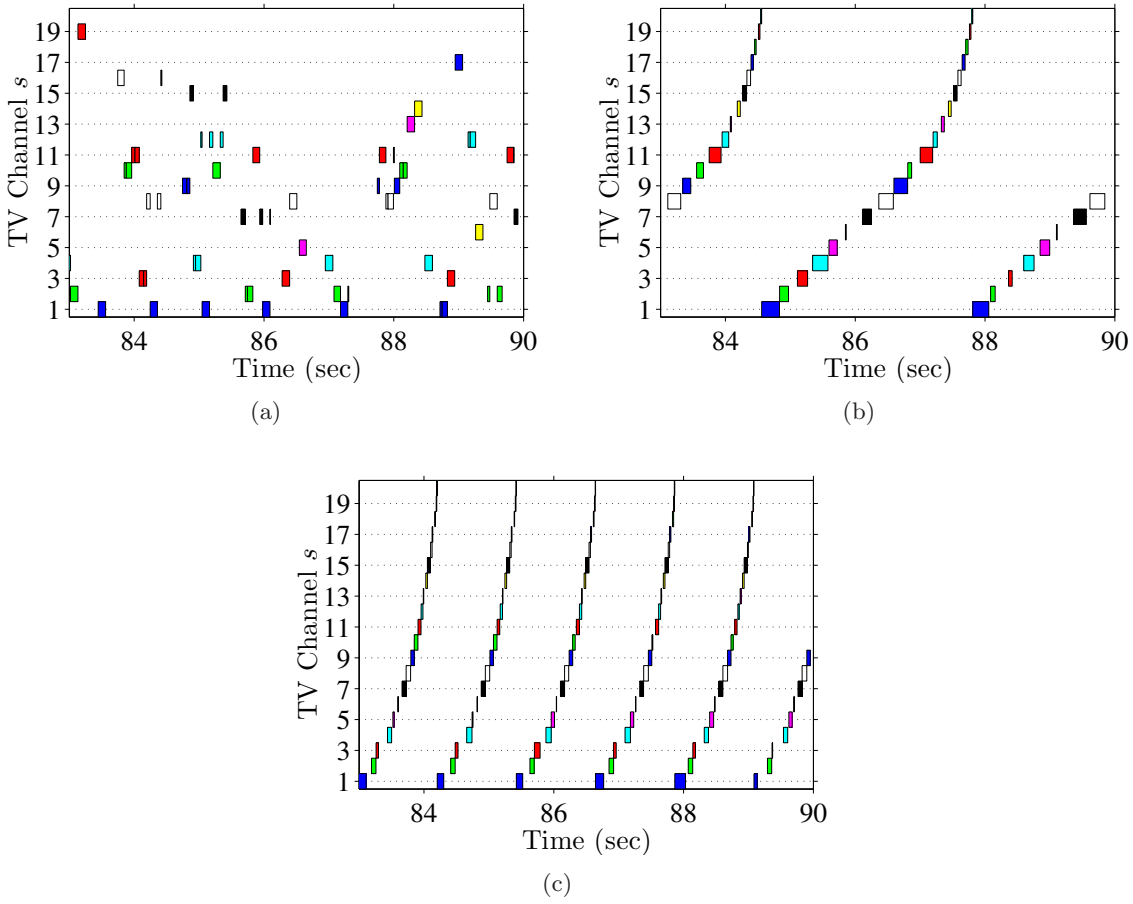


Figure 4.7: Burst schedules produced by considered algorithm: (a) SMS, (b) $\text{VBR}_{70\%}$, and (c) RVBR_1.

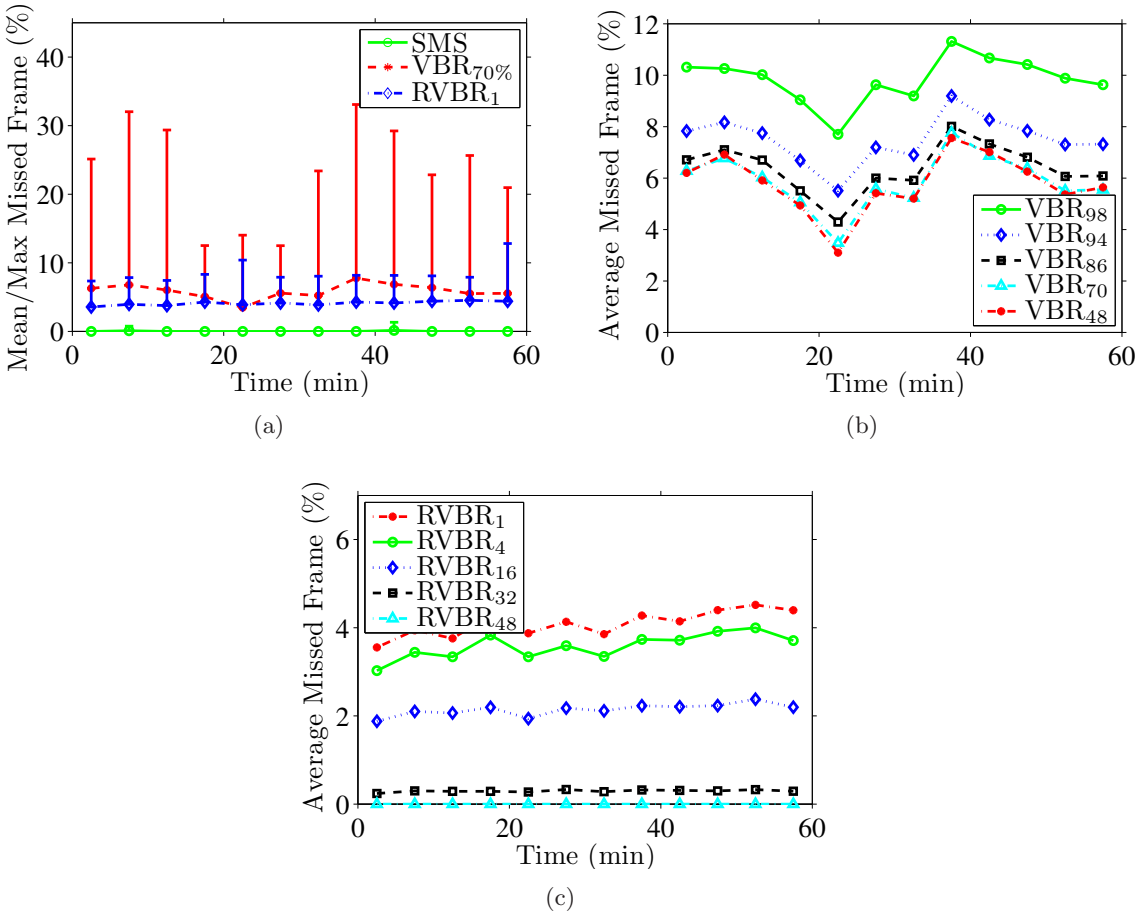


Figure 4.8: Missed frame ratio produced by: (a) all considered algorithms, (b) the VBR_α algorithm with various α values, and (c) the RVBR_β algorithm with various β values.

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 [67]. These trace files are for CIF video streams coded by H.264/AVC coders at 30 fps. We follow the recommendations given in [106] 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 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 above. We instruct the simulator to jointly encode 10 VBR

video streams, based on the video traces from a Video Trace Library [67]. The simulator employs Lagrangian optimization method [68] 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$_\tau$ algorithm, and we vary τ value from 1 to 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.

Last, we evaluate the SMS$'$ algorithm in open-loop networks. We do not consider the SMS$'$ algorithm in closed-loop networks as its performance would be *the same* as the SMS algorithm. For open-loop networks, we use the same broadcast parameters and concurrently broadcast 20 VBR video streams as mentioned above. We use the SMS$'$ algorithm to broadcast these video streams several times with different lookahead window size ζ: from 0 to 70 secs. For each ζ value, we compute the total number of missed frames among all videos. We also calculate the system-wide energy saving.

4.6.2 Simulation Results

Visual Validation. We first plot the burst schedules computed by each considered algorithms in Figure 4.7 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, e.g., the air medium is idle around the time 86 sec in Figure 4.7(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 4.7(c), have very short bursts, which lead to low energy saving. Therefore, current scheduling algorithms are not optimal in terms of energy saving either. Our results illustrate that the current scheduling algorithms are not efficient in terms of goodput and energy saving.

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 4.8(a), which shows that the SMS algorithm produces almost no missed frames, while

$\text{VBR}_{70\%}$ 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 $\text{VBR}_{70\%}$ 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 Figure 4.8(a) it produces almost no missed frames. We plot the results of VBR_α algorithm with different α values in Figure 4.8(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_α algorithm, no burst schedule with acceptable QoS is possible. Then, we plot results of the RVBR_β algorithm with various β values in Figure 4.8(c). This figure shows that the average missed frame ratio decreases when the preroll delay of the RVBR_β 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_β 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 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 4.6.1, and we compute the missed frame ratio at each step. We plot the average missed frame ratio throughout the broadcasts in Figure 4.9(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 $\text{VBR}_{70\%}$ algorithm can only broadcast 2 video streams. In Figure 4.9(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_α algorithm, it can only broadcast 2 video streams. Moreover, a β value larger than 16 is required for the RVBR_β 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

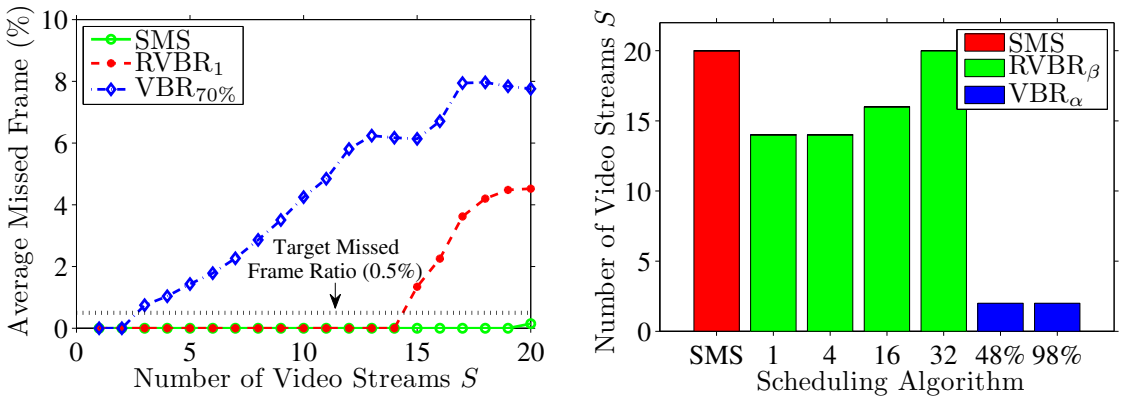


Figure 4.9: (a) Missed frame ratio achieved by various scheduling algorithms with different number of video streams. (b) Maximum number of video streams that can be broadcast.

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 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 4.10; 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 $\text{VBR}_{98\%}$ and RVBR_{16} with a margin as high as 12% and 5%, respectively. This experiment shows

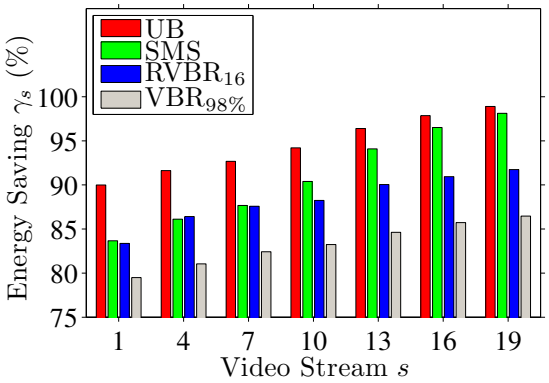


Figure 4.10: Energy saving achieved by considered burst scheduling algorithms and a conservative upper bound.

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 4.11. We draw two observations on this figure. First, the SMS algorithm constantly results in higher energy saving than the current algorithm. For example, Figure 4.11(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 4.11(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 4.12. 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.

Lookahead Window Size. We next study the benefits of using the SMS$'$ algorithm in open-loop networks. We quantify the performance different between the SMS algorithm

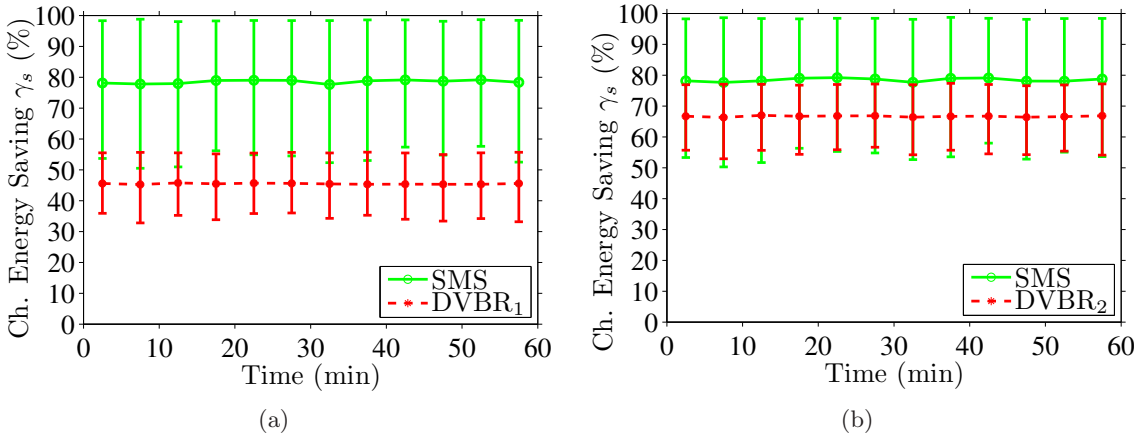


Figure 4.11: Per-channel energy saving comparison between the SMS and DVBR$_\tau$ algorithms, with: (a) $\tau = 1$ and (b) $\tau = 2$.

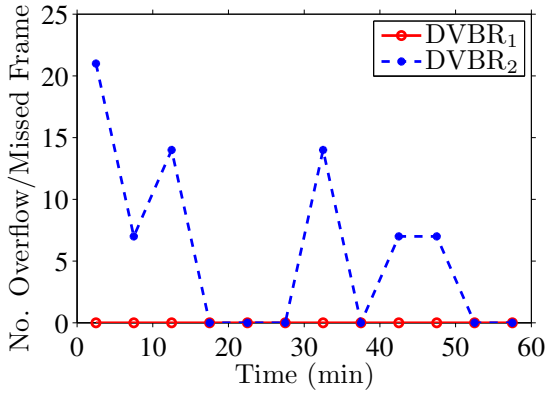


Figure 4.12: DVBR$_2$ results in overflow/missed frames on mobile devices.

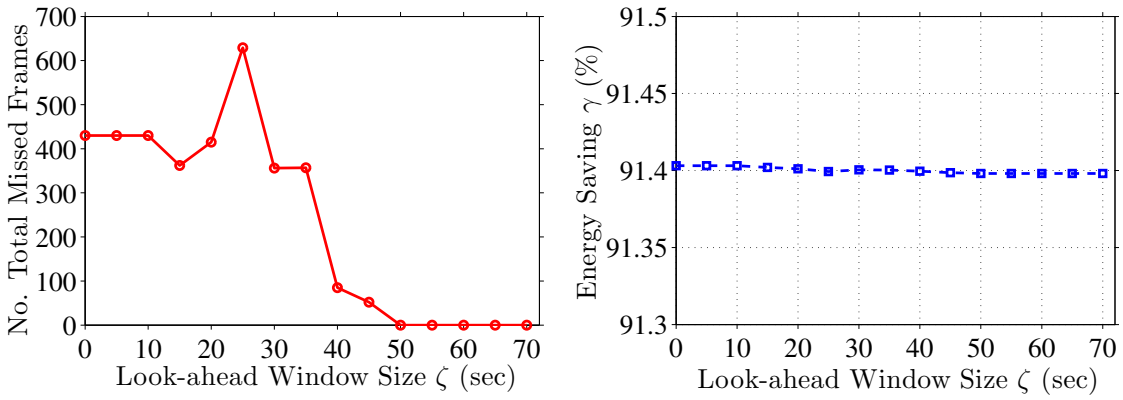


Figure 4.13: Implication of lookahead window size on: (a) number of late frames and (b) energy saving.

and the SMS′ algorithm with different ζ values, and we plot the results in Figure 4.13. We note that SMS algorithm is essentially the same as the SMS′ algorithm with $\zeta = 0$. We first run each considered algorithm and compute the total number of late frames of all 20 video streams throughout the 60 min broadcast. We plot the results in Figure 4.13(a). We draw two observations on this figure. First, larger ζ value in general results in fewer missed frames. For example, with 50 sec (or longer) lookahead window, the SMS′ algorithm leads to no missed frames by transmitting some video data earlier when the broadcast network had some slack times. Second, we notice that, the number of missed frames is not monotonically decreasing as ζ increases. For example, the SMS′ algorithm performs worse with $\zeta = 25$ than $\zeta = 20$. This happens because the SMS′ only considers a small lookahead window, and the way it reuses slack time (line 13 in Figure 4.6) may not be *optimal* in terms of minimizing buffer violation instances. This, however, is understandable as we have shown that the burst scheduling problem is NP-complete. Network operators who have a lookahead window of $\hat{\zeta}$ sec, may run the SMS′ multiple times with several $\zeta < \hat{\zeta}$ values, and then pick the best burst schedule out of all returned ones. In summary, Figure 4.13(a) indicates the SMS′ algorithm effectively reduces the number of missed frames.

We next consider the energy saving achieved by the SMS′ algorithm. We compute the system-wise energy saving γ for the SMS′ algorithm with different ζ value, and we plot them in Figure 4.13(b). This figure clearly shows that energy saving degradation caused by larger ζ is negligible: the energy saving is about 91.4% in all experiments. Hence, the SMS′

algorithm produce glitch-free burst schedules without sacrificing energy saving.

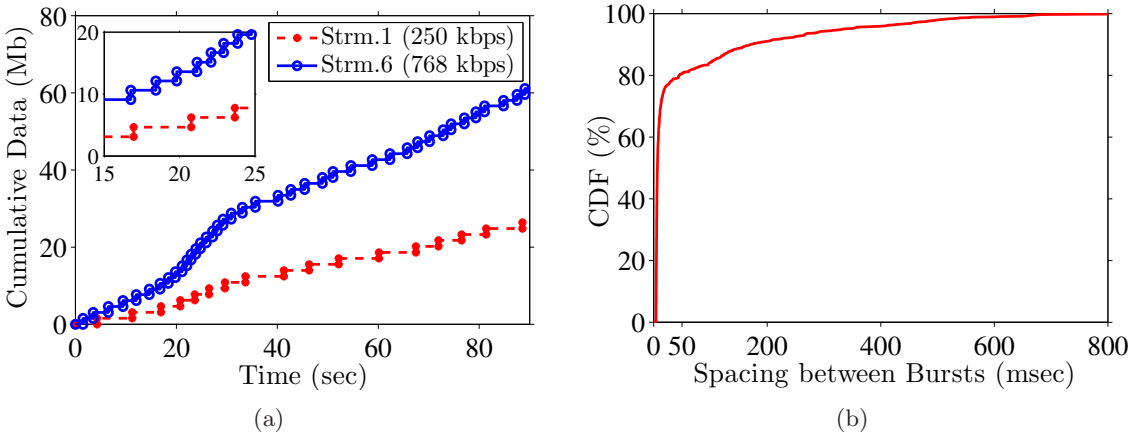
Running Time. Finally, we report the running time of the SMS algorithm on a commodity PC with a 2.33 GHz processor and runs Linux. It takes the proposed algorithms less than 1 sec to construct the burst schedule for the 60 min simulation. This clearly shows that the proposed algorithms incur negligible processing overhead, and can run in real time.

4.7 Implementation on a Mobile TV Testbed

In this section, we use a mobile TV testbed implemented in our Lab to evaluate the SMS algorithm. This testbed, described in Section 3.6, complies with the DVB-H standard [9,10].

4.7.1 Testbed Setup

We have implemented the proposed SMS algorithm in a complete testbed for mobile TV networks, which is detailed in Section 3.6. 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 [16]. 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 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 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) (b)

Figure 4.14: (a) Buffer dynamics for the SMS algorithm, and (b) time spacing between successive bursts.

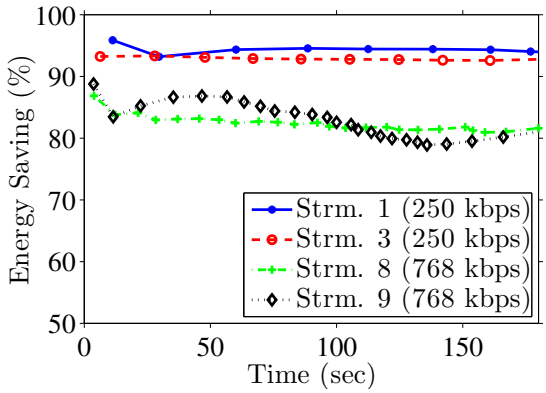


Figure 4.15: Energy saving achieved by our algorithm for individual video streams.

4.7.2 Results from Mobile TV Testbed

Correctness. 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 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 4.14(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 burst 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.

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 4.14(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 4.15 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.

Running Time. 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 (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.

4.8 Conclusions

We studied the problem of broadcasting multiple 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 glitch-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 lookahead window, and can handle the dynamic nature of the video broadcast service. However, in open-loop networks, the SMS algorithm might lead to some playout glitches when the aggregate bit rate of all video streams exceeds the network bandwidth. We propose another scheduling algorithm, called SMS$'$ to address this problem. The SMS$'$ algorithm employs a longer lookahead window and utilize any slack times of the air medium to absorb the burstiness in aggregate bit rate. The length of this lookahead window is a configurable system parameter. The SMS$'$ algorithm also runs fast, and thus is an online algorithm as well.

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. Finally, we study the implication of different lookahead window size in the SMS$'$ algorithm. The simulation results reveal that: (i) longer lookahead windows in general lead to fewer missed frames and (ii) lookahead window sizes have little impact on energy saving of mobile devices.

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 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.

Chapter 5

Controlling Channel Switching Delay

In this chapter, we solve the problem of controlling channel switching delay in mobile video broadcast networks. We first show that existing base stations cannot efficiently control channel switching delay. We then propose three time slicing schemes to ensure that switching from a channel to any other channel never exceed a given maximum switching delay. We implement the proposed schemes in a real mobile TV testbed to show their practicality and efficiency.

5.1 Introduction

We study Problem 3 to construct burst schedules for controlling channel switching delays. While time slicing enables mobile devices to save energy, it increases the channel switching delay, which is the time that a user waits before s/he starts viewing a selected channel when a change of channel is requested by that user. The switching delay is an important performance metric, because many users quickly flip through several channels before they decide on watching the specific ones. Operators of mobile video broadcast networks have to maintain low and constant switching delays, as long and variable switching delays are annoying to users and may turn them away from the mobile video broadcast service. Our goal is to design time slicing schemes to bound the maximum switching delay from a channel to any other channel to a target value *without* sacrificing the energy saving of mobile devices.

We first analyze the time slicing scheme currently used in many deployed mobile video broadcast networks, and we show that it cannot efficiently achieve short channel switching delays. More precisely, current time slicing scheme achieves low energy saving when short channel switching delays are required. We then propose three new time slicing schemes that ensure that a given maximum switching delay is not exceeded, while at the same time the energy consumption of mobile devices is minimized. Our proposed schemes employ simulcasting of video streams with and without scalable video coding techniques, which was described in Section 2.2.2. We prove the correctness of the proposed schemes and derive closed-form equations for the achieved energy saving. We numerically analyze the performance of the proposed schemes and provide guidelines for network operators to choose the most suitable time slicing scheme for their mobile video broadcast networks. We implement the proposed schemes in a mobile TV testbed. Our experimental results validate our theoretical analysis and show that the proposed schemes indeed meet the target channel switching delays and achieve high energy savings for mobile devices: up to 95% energy saving is observed.

5.2 Related Work

Channel switching delay is composed of several parts, in which frame refresh delay and time slicing delay are the two dominating contributors [29, 33]. The frame refresh delay refers to the time period between receiving the first bit of a new video stream and receiving the next random access point, typically an intra-coded frame, of that video. The time slicing delay refers to the time period between locking onto a mobile TV signal and reaching the first burst of the selected TV channel. Existing switching delay reduction solutions in the literature can be roughly categorized into three classes: solutions that use an auxiliary cellular network [32], solutions that reduce frame refresh delays [29, 33, 75–77, 107], and solutions that reduce time slicing delays [16, 75]. We briefly survey each of them in the following.

Ollikainen and Peng [32] propose a vertical handover approach for DVB-H networks, where each mobile device maintains a unicast connection to the base station over a Universal Mobile Telecommunications Systems (UMTS) network. Video streams are not only carried by the DVB-H network but also relayed by the cellular network, which enables a mobile TV device to quickly handoff to the unicast connection when the DVB-H signal is degraded. This

auxiliary network can also be used for reducing channel switching delays. Unfortunately, maintaining a unicast connection for each mobile device for video traffic imposes tremendous load on cellular networks and streaming servers, and therefore is not scalable. In contrast, our solution requires no additional connections over the cellular network and is simpler and more scalable.

To reduce frame refresh delays, Vadakital et al. [75] propose to periodically add redundant intra-coded frames into video streams coded by H.264/AVC. In H.264/AVC, redundant frames are only decoded when primary, normal, frames are not decodable. By frequently adding low quality intra-coded redundant frames into a video stream, more random access points are added, which in turn reduces refresh delays. Instead of sending low-quality intra-coded frames over dedicated channels, intra-coded frames can be dynamically inserted at the beginning of every burst by the IP encapsulator to shorten frame refresh delays [29,33,76,77]. This is done by sending two coded video streams, primary and auxiliary, from the video streaming server to the IP encapsulator over a local wired network. The IP encapsulator substitutes an intra-coded frame (on the auxiliary stream) for an inter-coded frame (on the primary stream) that is about to be encapsulated as the first frame in a burst. Since every burst starts with an intra-coded frame, a mobile device can decode immediately after receiving the first frame in each burst. The works on minimizing refresh delays are orthogonal to our work, and they can be combined with our work on reducing the time slicing delay.

The closest works to ours are those that try to reduce the time slicing delay [16,75]. The DVB-H standard suggests to employ parallel elementary streams for channel switching delay reduction [16], in which network operators bundle several TV channels into a channel group. The channel group is then encapsulated into a series of time slicing bursts. Mobile devices which want to receive any of these TV channels will have to process all bursts of the group, despite the fact that not all the data is useful to them. In addition, parallel elementary streams cannot reduce channel switching delay if the selected TV channel is not in the same channel group as the current TV channel. Therefore, determining the channel grouping strategy itself is a difficult question. In contrast, our proposed schemes provide guaranteed switching delays between any two arbitrary channels, and do not need any heuristics to group TV channels. Since the channel grouping strategy is not specified in [16], we cannot compare our work against the parallel elementary stream method. Even if the grouping strategy is known, it would be an unfair comparison (in favor of our schemes), because our schemes are designed to minimize energy consumption, while the parallel elementary

Table 5.1: List of symbols used in controlling channel switching delays.

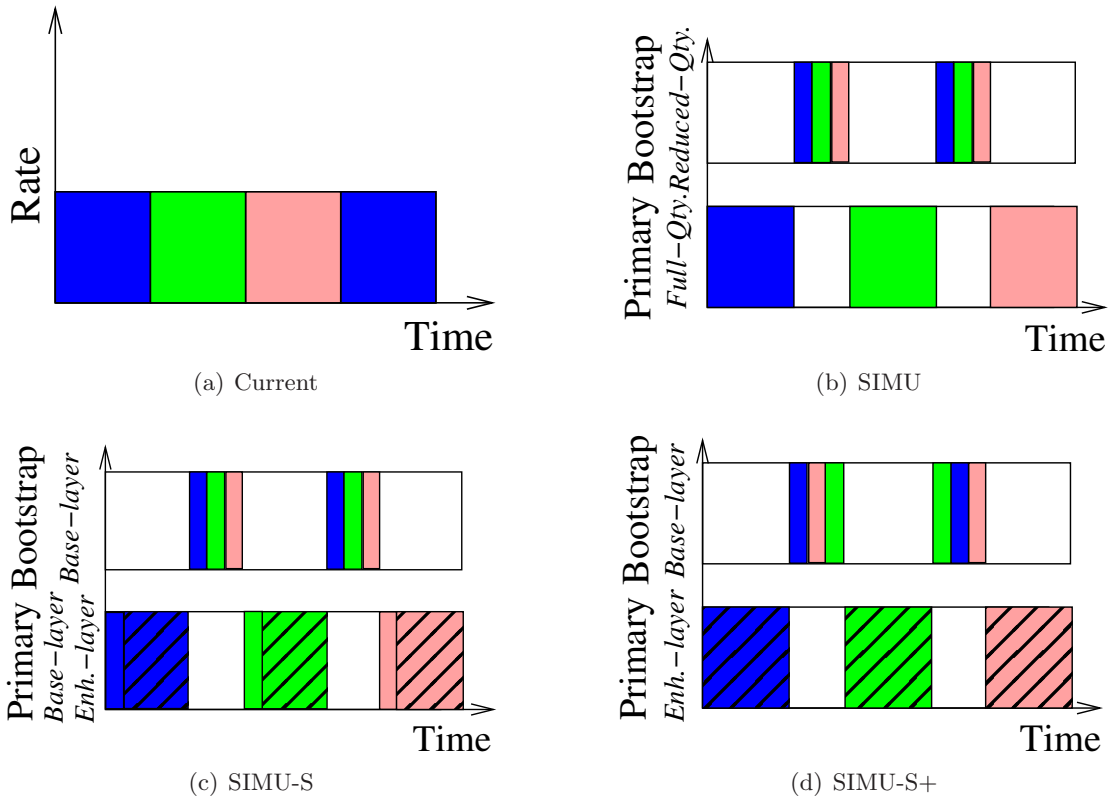
Sym	Definition	Sym	Definition
T_o	overhead duration	d	channel switching delay
S	number of video streams	d_m	maximum switching delay
R	burst bit rate	γ	energy saving
b	receiver buffer size	γ_b	γ in bootstrap stage
r	channel bit rate	γ_p	γ in steady stage
r_l	reduced-quality bit rate	w	per-channel watch time

streams method results in suboptimal energy consumption since mobile devices spend more time and energy on receiving irrelevant data. Hence, we compare against the time slicing scheme where each mobile device *only* receives the video data of its TV channel. This is referred to as the current time slicing scheme in this chapter, because it is currently used in deployed networks.

Vadakital et al. [75] consider the problem of interleaving C versions of a video stream to minimize the time slicing delay for C classes of mobile devices, where class-c mobile devices can decode any stream version i ($i \leq c$). They show that arranging the bursts of these versions maximally apart from each other minimizes the time slicing delay. Unlike their work, which does not bound the time slicing delay, we propose a systematic way to meet the (controllable) delay requirement. More importantly, their work fixes the inter-burst durations of different stream versions, hence allocates too many bursts for versions at low bit rates, which results in poor energy savings. In contrast, our allocation schemes are provably optimal in terms of energy saving. We do not compare our schemes against their work in this chapter.

5.3 Problem Statement and Solution Approach

In this section, we formally state Problem 3 addressed in this chapter, and we analyze the performance of the solution currently used in many deployed mobile video broadcast networks. We list all symbols used in the chapter in Table 5.1 for quick reference. We also present an overview of our proposed solutions and discuss the advantages and disadvantages of each of them.



(a) Current (b) SIMU

(c) SIMU-S (d) SIMU-S+

Figure 5.1: Time slicing schemes in mobile video broadcast networks. Bursts of three TV channels are shown with different colors. (a) Current scheme, and (b)–(d) proposed schemes, which use simulcast with and without scalable video coding. For scalable video streams, hatched areas represent the enhancement layers.

5.3.1 Problem Statement

We consider a mobile video broadcast network in which a base station concurrently broadcasts several TV channels to a large number of mobile devices over a shared air medium with bandwidth R kbps. Each TV channel carries a video stream and is allocated a bit rate r kbps. r is much smaller than R. The base station broadcasts each TV channel in a series of *bursts* at bit rate R kbps, which is referred to as a *burst train*. The burst size is denoted by b kb. A mobile receiver receives a burst of data and turns off its receiving circuit until the next burst of the same burst train. This is called time slicing. While time slicing saves the energy of the mobile devices, it may increase the channel switching delay. Figure 5.1(a) illustrates a simple example in which a base station broadcasts bursts of three TV channels (shown with different colors in the figure). Mobile devices tuned-in for TV channel 1 (dark bursts) turn their receiving circuits off during the transmission of the other bursts. If a user watching TV channel 1 decides to switch to channel 3, the user will have to wait until the next burst of channel 3 is broadcast by the base station. This is referred to as the time slicing delay. As mentioned in Section 1.2.3, in addition to the time slicing delay, the user will have to wait for other delays before viewing channel 3, such as the delay until the first intra-coded video frame arrives (known as frame refresh delay) and the time to decode and render the stream. These delays combined are referred to as the channel switching delay, which is denoted by d. Since the time slicing delay is the major component of the channel switching delay, we focus in this chapter on designing efficient time slicing schemes to control the time slicing delay. We assume that other delays are fixed, and we do not consider them anymore.

We notice that the relative start time of each burst is recorded in the header of its predecessor burst such that the receivers know when they need to wake up to receive data [8, 10, 11]. As the start time is sent in relative form, its accuracy is not affected by any constant delays between the base station and its receivers. However, the start time is sensitive to the clock jitter caused by the inaccuracy of the timers of mobile devices. To cope with this, mobile devices have to wake up their receiving circuits slightly earlier to absorb the clock inaccuracy, which is referred to as *delay jitter*, and it is in the order of 10 msec [16]. Furthermore, waking up the receiving circuits is not instantaneous, because it takes some time to lock to the frequency and synchronize to the symbols before data can be demodulated. This is called *resynchronization time*. We define the *overhead duration* as

the amount of time the receiving circuits of mobile TV devices must be turned on before the burst time for successful demodulation, which is the sum of the delay jitter and the resynchronization time. We denote the overhead duration by T_o. T_o is in the range of 50—250 msec [9,10,16], and in Section 2.6, we empirically show that a recent Nokia cellular phone has T_o in the range of 80–140 msec. Therefore, we use $T_o = 100$ msec in our analysis below, if not otherwise specified.

The energy saved by a mobile device because of the time slicing scheme is denoted by γ, and it is calculated as the ratio of time the receiving circuit is in off mode to the total time [16, 19]. Since TV channels have the same bit rate r, their bursts have the same size and are periodically broadcast by the base station. Thus, the energy saving γ achieved by any mobile device is the same regardless of the specific TV channel being received by that device.

With the above definitions and notations, we can state Problem 3 as designing optimal time slicing schemes for a broadcast network with bandwidth R kbps, such that the channel switching delay from any channel to any other channel is at most d_m sec, where d_m is given as the maximum allowed channel switching delay. Solving this problem is important for the success of mobile TV services, and it has direct impacts on the profitability of service providers. This is because high switching delays may drive subscribers away from mobile TV services. In addition, this problem maximizes the energy saving for mobile devices, thus stretches the possible viewing time for mobile users and allows them to consume more TV content. Therefore, optimally solving the problem is beneficial for both operators and users of mobile TV services.

5.3.2 Limitations of the Current Time Slicing Scheme

Current mobile video broadcast networks implement simple time slicing schemes. For example, the scheme proposed in the DVB-H standard documents [16, pp. 66] provides schedules for one TV channel: it allocates a new burst only after the data of its preceding burst is consumed by the player at the receiver. This means that bursts of all TV channels will be of the same size b, because the TV channels are encoded at the same bit rate r. While the current time slicing scheme simplifies the design of the base station, it may lead to long switching delays and/or waste the energy of mobile devices. We analyze the channel switching delay d and energy saving γ of this scheme in the following. Since mobile video broadcast networks allocate periodical bursts to keep receivers in a *steady* state without

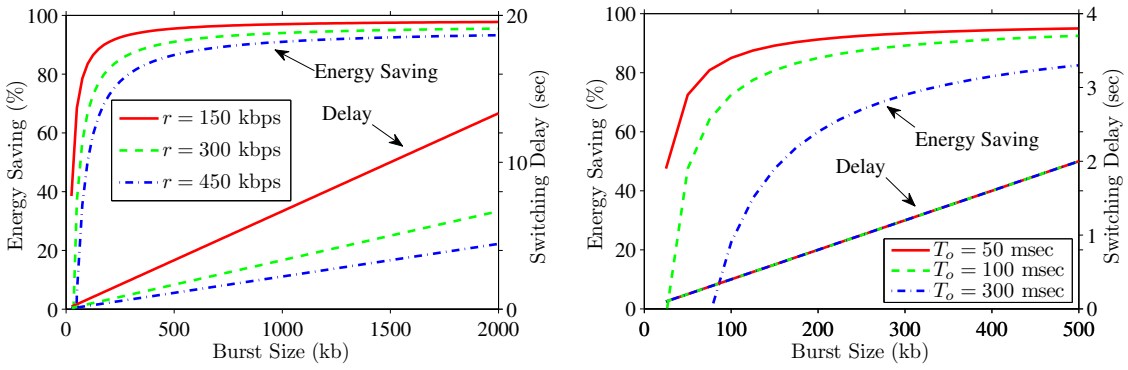


Figure 5.2: The trade-off between energy saving and switching delay with different: (a) video bit rates, and (b) overhead values.

buffer over/underflow, the inter-burst period between two adjacent bursts of the same TV channel should be computed so that the number of received bits is equal to the number of consumed bits during that period. Given that each burst has a size b and the TV channel bit rate is r, the inter-burst period should be b/r to avoid buffer over/underflow. Thus, the worst case channel switching delay is $d = b/r$.

For the energy saving, consider a burst period b/r between two adjacent bursts. Since we need to transmit b kb over a shared medium with R kbps bandwidth, the burst duration is b/R sec. We also need to take into consideration the overhead duration T_o, during which the receiving circuits of the receivers have to be open to search for and lock on the radio signal. Thus, we have:

$$\gamma = 1 - \frac{b/R + T_o}{b/r} = 1 - \frac{r}{R} - \frac{T_o r}{b}, \tag{5.1}$$

where $b/R + T_o$ is the on-time of the receiving circuit and b/r is the total time. Eq. (5.1) reveals an important trade-off between the channel switching delay and the energy saving as both d and γ increase when b increases. To illustrate this trade-off, we vary the burst size b between 10 kb and 2000 kb and compute the energy saving γ and the channel switching delay d. We set $R = 10$ Mbps and $T_o = 100$ msec. Figure 5.2(a) plots γ and d versus b for three sample TV channel bit rates: 150, 300, 450 kbps. Notice that the figure has two y-axes. This figure shows that the channel switching delay can be as high as 13 seconds for TV channels at low bit rates.

The current time slicing scheme can ensure a given maximum switching delay d_m by

scaling down the burst size, i.e., by setting $b = rd_m$. To examine the implication of scaling b down, we plot the trade-off again with different overhead T_o values in Figure 5.2(b), where r is fixed at 250 kbps. This figure shows that there is a dramatic decrease in energy saving when a small switching delay is desired for all practical T_o values. For example, for $T_o = 100$ msec, reducing the channel switching delay from 1.5 to 0.25 seconds results in reducing the energy saving from 90% to 55%. This shows that reducing the switching delay by reducing the amount of data transmitted in each burst is not efficient in terms of energy saving, and a better solution for Problem 3 is needed.

5.3.3 Overview of the Proposed Solutions

To achieve both low switching delays and high energy savings, we propose to *simulcast* each TV channel over two burst trains. Simulcast refers to a strategy that simultaneously broadcasts two versions of the same TV channel, but at different bit rates. We design new time slicing schemes such that one burst train is optimized for energy savings (referred to as the *primary* train), and the other burst train is optimized for switching delays (referred to as the *bootstrap* train). A mobile device that just switches to a new TV channel tunes to the bootstrap train first, which enables the device to start the playout of video very quickly. The device tunes to the primary train upon the next primary burst of its TV channel is transmitted. Switching over to the primary train enables the device to save more energy. Devices that are receiving the bootstrap trains are referred to as in the bootstrap stage, and devices that are receiving the primary trains are considered in the steady stage.

We further propose and analyze two methods for simulcast: (i) simulcast with traditional nonscalable video coding (referred to as SIMU), and (ii) simulcast with scalable video coding (referred to as SIMU-S). In its simplest form, SIMU broadcasts the same video stream twice with different burst sizes over the bootstrap and primary burst trains. This, however, reduces the effective utilization of the air medium as fewer TV channels can be broadcast. To mitigate this problem, a reduced-quality video stream can be transmitted over the bootstrap burst train, while the full-quality stream is transmitted over the primary burst train. Since mobile TV receivers only playout bursts from bootstrap trains for a short time period (order of seconds), showing a lower quality video in the bootstrap stage is not very noticeable. In Section 5.5.2, we experimentally show through encoding multiple video sequences using an H.264/SVC coder at reduced bit rates that the quality is reasonable. Furthermore, users who quickly flip through TV channels barely realize that the video quality is reduced, because

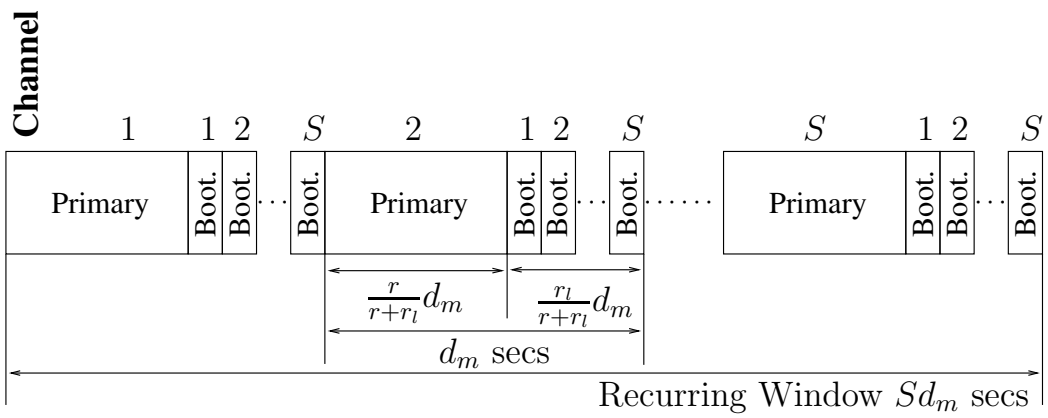


Figure 5.3: Burst allocation for the SIMU and SIMU-S schemes.

it takes human eyes some time to detect the visual details in a new scene/channel. In fact, a recent work [108] shows that users prefer a shorter reduced-quality startup phase than a longer one, which indicates that reduced-quality video streams are not annoying to users. Finally, the video quality in the bootstrap stage is a *configurable* parameter, and can be determined by service providers. Figure 5.1(b) illustrates the SIMU scheme. The SIMU scheme, while effective in delay reduction and energy saving, has two drawbacks. First, for each video, two separate streams need to be compressed and stored, which imposes higher costs on computational power and storage space. Second, switching from the bootstrap stream to the primary stream introduces complexities such as stream synchronization and reference frame management.

To address these drawbacks, SIMU-S employs scalable video coding, where a video stream is encoded into a base layer and one or more enhancement layers. SIMU-S transmits the reduced-quality stream (base layer) over the bootstrap burst train, and the full-quality stream (both base and enhancement layers) over the primary burst train. SIMU-S requires lower computational power and smaller storage space, simplifies video stream switching procedure, and allows gradual quality transition from reduced- to full-quality videos. Figure 5.1(c) illustrates the SIMU-S scheme. While SIMU-S addresses the storage and multiple compression problems, it still incurs replication over the air medium. As shown in Figure 5.1(c), the base layers are broadcast over both the primary and bootstrap trains. This replication may not be desirable in broadcast networks with limited bandwidth. To handle this case, we propose to broadcast the base layer only over the bootstrap burst train and

the enhancement layer over the primary burst train. We call this scheme SIMU-S+, and it is shown in Figure 5.1(d).

As we will show in the next section, all three proposed schemes (SIMU, SIMU-S, and SIMU-S+) can provide a guarantee on channel switching delay, i.e., they solve Problem 3. However, they have different advantages and are suitable in different environments. SIMU is useful in legacy environments where scalable video decoders are not widely available to mobile devices. SIMU-S is more suitable for mobile video broadcast networks with some idle bandwidth, while SIMU-S+ is for bandwidth saturated mobile video broadcast networks. We note that SIMU and SIMU-S result in much better energy savings than SIMU-S+, but lower bandwidth utilization.

5.4 Details of the Proposed Solutions for Bounding Switching Delays

In this section, we present the detailed design of the proposed time slicing schemes. We also derive closed-form equations for the achieved energy saving.

5.4.1 Time Slicing with Simulcast

As shown in Figure 5.1(b), SIMU transmits two versions of each video stream: one over the primary burst train with bit rate r, and another over the bootstrap burst train with bit rate r_l, where $r_l \leq r$. We propose an optimal (in terms of energy consumption) time slicing scheme that specifies the transmission time and the size of each burst in the primary and bootstrap trains. This time slicing scheme is illustrated in Figure 5.3. The basic idea is to divide the time into recurring windows of size Sd_m each, where S is the number of TV channels and d_m is the maximum allowed switching delay. Within this window, only one (large) primary burst of each TV channel is transmitted. This burst has just enough video data to be played back until the next primary burst of the same TV channel. The primary bursts of TV channel $s (s = 1, 2, \ldots, S)$ are allocated as follows:

$$< (s-1)d_m, \;\; Sd_m r >, \tag{5.2}$$

where the first element is the burst start time in seconds and the second element is the burst size in kb. Allocating the primary bursts according to Eq. (5.2) minimizes the energy

consumption in the steady stages (as shown below in Theorem 10), but it does not ensure that the switching delay is bounded by d_m. To do so, the inter-burst period between two bursts of the same TV channel in the bootstrap train must be at most d_m. Thus in a window of size Sd_m, we need S bootstrap bursts for each TV channel, which are equally spaced with distance d_m. The bootstrap bursts of TV channel $s(s = 1, 2, \ldots, S)$ are allocated as follows:

$$< (k-1)d_m + \frac{r}{r+r_l}d_m + (s-1)\frac{r_l}{r+r_l}\frac{d_m}{S},\ \ d_m r_l >, \qquad k = 1, 2, \ldots, S, \tag{5.3}$$

where the first element is again the burst start time, and the second is its size. The following theorem shows that the above allocation for primary and bootstrap bursts is valid and optimal.

Theorem 10. *For simulcasting with nonscalable coding in mobile video broadcast networks, allocating the primary and bootstrap bursts according to Eqs. (5.2) and (5.3), respectively yields a valid time slicing scheme (i.e., with no buffer over/underflow for any TV channel), meets the switching delay constraint d_m, and maximizes the energy saving for mobile devices. Moreover, the energy saving is given by*

$$\gamma_b = 1 - \frac{r_l}{R} - \frac{T_o}{d_m}, \tag{5.4}$$

and

$$\gamma_p = 1 - \frac{r}{R} - \frac{T_o}{Sd_m}, \tag{5.5}$$

where γ_b and γ_p are energy savings for devices in bootstrap and primary (steady) stages, respectively.

Proof. Consider a mobile device receiving an arbitrary TV channel out of the S channels. If the mobile device is in the primary stage, it gets $Sd_m r$ kb video data in every recurring time window of length Sd_m sec. This is equivalent to streaming at rate $(Sd_m r)/(Sd_m)$ kbps, which is equal to the full-quality video at bit rate r. If the mobile device is in the bootstrap stage, it gets $d_m r_l$ kb video data at interval of d_m as there are S bursts assigned to each TV channel in each recurring window of length Sd_m sec. This is equivalent to streaming at rate $d_m r_l/d_m$, which is equal to the reduced-quality video at bit rate r_l. Since the received bit rates are equal to the consumed bit rates, our allocation leads to no buffer over/underflow and is a valid time slicing scheme.

Next, consider any two adjacent bootstrap bursts (k and $k+1$) for a TV channel s. According to Eq. (5.3), the difference in the start time between them is d_m, as $(k-1)d_m$ is

the only term that depends on k. Therefore, for a channel switching event occurring at an arbitrary time, there is a bootstrap burst within at most d_m sec. This shows that our time slicing scheme meets the switching delay constraint.

To show that our burst allocation scheme in SIMU minimizes energy consumption, we first show that any scheme that does not use simulcasting (i.e., broadcasts a *single* version of each video at bit rate r) will result in higher energy consumption than ours. To guarantee a maximum switching delay of d_m, any such allocation scheme *must* place two successive bursts of the same video no further apart than d_m sec. Therefore, it can achieve energy saving no better than $1 - \frac{r}{R} - \frac{T_o}{d_m}$, which is always worse than the energy saving of our scheme $\gamma_p = 1 - \frac{r}{R} - \frac{T_o}{Sd_m}$ (γ_p is derived later in this proof). In addition, simulcasting schemes with more than two versions of the video will always consume more energy than our scheme (which has only two versions). This is because more versions of each video results in smaller (thus more) bursts and incurs more overhead duration periods T_o.

Now we show that our allocation scheme minimizes energy consumption by contradiction. Since we consider a steady system where the receiving number of bits equals the consumed number of bits, any valid time slicing scheme with the same recurring window size consumes the same amount of energy on receiving video data. Therefore, what differentiates the energy consumption of one time slicing scheme from another is the number of overhead duration T_o periods incurred in receiving the data. This enables us to reduce our problem of minimizing energy consumption to minimizing the number of bursts and hence minimizing the number of overhead periods incurred. Assuming there exists a valid time slicing scheme with recurring window $w_1 > Sd_m$ and the same number of primary bursts as our scheme, i.e., the new scheme results in better energy consumption. To meet the delay constraint, the new scheme *must* allocate bootstraps burst for every TV channel at time intervals d_m. Moreover, to keep the number of primary bursts the same as our scheme, the new scheme can only assign one burst for each TV channel. Since the burst bit rate is R, it takes the new scheme $\frac{rw_1}{R} + \frac{r_l w_1}{R}$ sec to complete one primary burst and S bootstrap bursts. Since $S = \frac{R}{r+r_l}$, manipulating the above equation leads to:

$$\frac{rw_1}{R} + \frac{r_l w_1}{R} = (r + r_l)\frac{w_1}{R} > \frac{(r + rl)Sd_m}{R} > d_m.$$

This contradicts the assumption that the new scheme is a valid allocation. Next, assuming there is a valid time slicing scheme with recurring window $w_2 < Sd_m$ that produces higher energy savings than our scheme. Both this new scheme and our scheme consist of S primary

bursts, while the new scheme has to allocate w_2/d_m bootstrap bursts for each TV channel to meet the switching delay constraint. Consider each burst imposes a fixed overhead duration, the new scheme always results in worse energy savings than our scheme in the steady stage because the overhead ST_o is averaged over a shorter time period $w_2 < Sd_m$. Meanwhile, the energy saving in bootstrap state is the same in both schemes. This completes our proof of optimality (in terms of energy saving).

Last, we derive the energy saving in both steady and bootstrap stages. Consider a recurring window of Sd_m. In the bootstrap stage, S bursts is assigned to each TV channel where the aggregate burst length is $d_m r_l S/R$. Following the definition of energy saving, we have $\gamma_b = 1 - \frac{d_m r_l S/R + ST_o}{Sd_m}$. In the steady stage, a burst with length $Sd_m r/R$ is assigned to each TV channel. Thus, we have $\gamma_s = 1 - \frac{Sd_m r/R + T_o}{Sd_m}$. □

The above theorem gives the energy savings for mobile devices in bootstrap and primary stages, and reveals that mobile devices in primary stage achieve higher energy savings than those in bootstrap stage. Therefore, the energy saving achieved by a user is determined by the average time period that user would continuously watch a TV channel, which is referred to as average *watch time* w in this chapter. The w value captures the channel switching behavior of a user and allows us to derive the average energy saving γ for that user. For a user with $w \geq Sd_m$, the average energy saving can be computed by taking the weighted sum of energy savings in bootstrap and primary stages, where the bootstrap stage lasts for $Sd_m/2$ sec on average. Hence, we write the energy saving for a user with watch time w as:

$$\begin{aligned}\gamma =& \frac{Sd_m/2}{w}\left(1 - \frac{r_l}{R} - \frac{T_o}{d_m}\right) + \frac{w - Sd_m/2}{w}\left(1 - \frac{r}{R} - \frac{T_o}{Sd_m}\right) \\ =& \frac{Sd_m}{2w}\left[\frac{r - r_l}{R} - \frac{(S-1)T_o}{Sd_m}\right] + \left(1 - \frac{r}{R} - \frac{T_o}{Sd_m}\right). \end{aligned} \tag{5.6}$$

This equation shows that the energy saving γ approaches γ_p when the watch time w is large. This is because mobile devices receive primary bursts most of the time. Moreover, we observe that $\gamma = \gamma_b$ when $w < Sd_m$, as mobile devices do not get chance to switch to primary burst trains at all.

5.4.2 Time Slicing with Simulcast and Scalable Coding

We next present our SIMU-S scheme, which adopts scalable video coding to save computational complexity and storage space and to reduce stream switching complexity. As shown

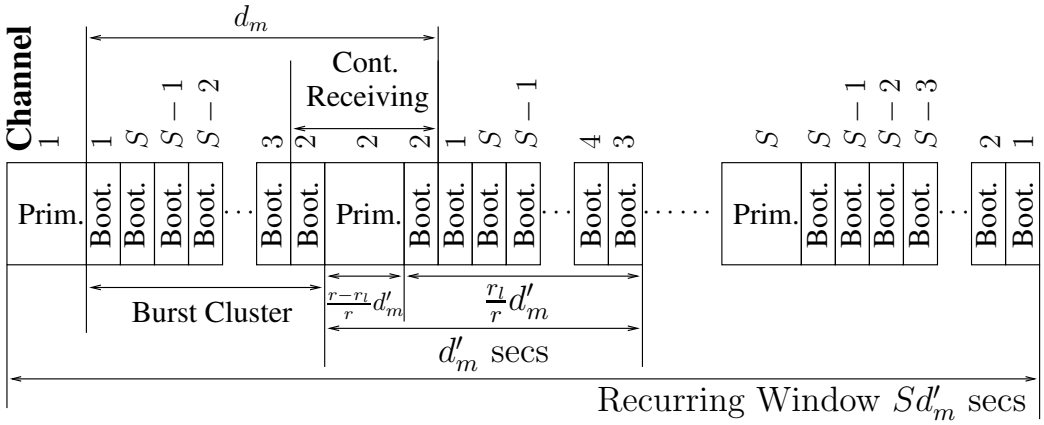


Figure 5.4: Burst allocation for the SIMU-S+ scheme.

in Figure 5.1(c), SIMU-S encodes each TV channel in two layers: base and enhancement layers. We provide a simple mapping between SIMU-S and SIMU, which will enable us to use the time slicing scheme presented in the previous subsection for SIMU-S. This mapping can easily be done by encoding each TV channel at base layer bit rate $r_b = r_l$ kbps and enhancement layer bit rate $r_e = r - r_b$ kbps. SIMU-S then streams the base layer over bootstrap bursts and the complete stream (both base and enhancement layers) over primary bursts using this mapping. Theorem 10 is also applicable to SIMU-S.

Next, we develop our SIMU-S+ scheme. Since bandwidth saturated networks cannot afford the replication of video data over the shared air medium, we cannot transmit base layer twice as we did in SIMU-S scheme. Therefore, as shown in Figure 5.1(d), we transmit the base layer only over bootstrap bursts and the enhancement layer over primary bursts. Mobile devices that just switch to a new TV channel can start playing the base layer for reduced-quality videos and add enhancement layer streams for full-quality videos whenever bursts on primary trains are available. We now design a time slicing scheme for SIMU-S+. The main difference between SIMU-S+ and SIMU-S is that mobile devices have to receive bootstrap bursts even in the steady stage. That is, more antenna on/off operations are required in each recurring window, which leads to higher energy consumption as each antenna on/off operation imposes an overhead duration T_o. To minimize this negative impact, we propose to shift every subsequent bootstrap burst by one as indicated in Figure 5.4. The shifting operation enables us to concatenate three bursts that belong to the same TV channel, say channel 2 as illustrated in Figure 5.4, together so that mobile devices can receive three

bursts without turning off their antennas.

However, due to the shifting operations, the last burst in each bootstrap burst cluster should be smaller than other bootstrap bursts. This is because the last burst has a shorter inter-burst time as it will become the first burst in the following bootstrap burst cluster, while all other bursts must be shifted to the right for one time slot. Let x be this bootstrap time slot width. Let d'_m be the time length of a primary burst and a cluster of bootstrap bursts. To guarantee that the channel switching delay does not exceed d_m sec, we set $d'_m + x = d_m$, which keeps any two adjacent bootstrap bursts of the same TV channel apart no longer than d_m sec. Since the bit rates of the primary and the bootstrap burst trains are given and the air medium bandwidth is a constant, we allocate $\frac{r-r_l}{r}d'_m$ time slots to each primary burst for any channel s, and $\frac{r_l}{r}d'_m$ for every bootstrap burst cluster. The latter one is then split into $\frac{r_l}{rS}\Big[d'_m - (S-1)x\Big]$ for the last bootstrap burst and $\frac{r_l}{rS}(d'_m + x)$ for others. Following the definition of x we have:

$$\frac{r_l}{rS}(d'_m + x) = x \implies x = \frac{r_l}{rS}d_m \text{ and } d_m = \frac{rS}{rS - r_l}d'_m.$$

The above equation gives us the bootstrap slot size x. We then allocate the primary bursts of TV channel $s (s = 1, 2, \ldots, S)$ as follows:

$$< (s-1)d'_m, \;\; (r - r_l)Sd'_m >, \tag{5.7}$$

where the first element is the burst start time in seconds, the second element is the burst size in kb, and $d'_m = \frac{rS-r_l}{rS}d_m$. The bootstrap bursts of TV channel s $(s = 1, 2, \ldots, S)$ are allocated as follows:

$$\begin{aligned} &< kd'_m - \frac{r_l d'_m}{r} + \frac{(i-1)r_l d_m}{rS}, \;\; \frac{r - r_l}{r}d_m r_l >, \;\; (1 \le k \le S \text{ and } i = S), \\ &< kd'_m - \frac{r_l d'_m}{r} + \frac{(i-1)r_l d_m}{rS}, \;\; d_m r_l >, \;\; (1 \le k \le S \text{ and } i \ne S), \end{aligned} \tag{5.8}$$

where the first element is the burst start time, the second is its size, and $i = \Big[(S + k - s) \mod S\Big] + 1$. Note that, i represents the position of each bootstrap burst within a burst cluster and accommodates the shifting operations. The following theorem shows that the above allocation for primary and bootstrap bursts is valid and optimal.

Theorem 11. *For simulcasting with scalable coding in mobile video broadcast networks, allocating the primary and bootstrap bursts according to Eqs. (5.7) and (5.8), respectively*

yields a valid time slicing scheme (i.e., with no buffer over/underflow for any TV channel), meets the switching delay constraint, and minimizes the energy consumption. Moreover, the energy saving is given by $\gamma = 1 - \frac{r}{R} - \frac{(S-1)T_o}{Sd'_m}$*, where* $d'_m = \frac{rS-r_l}{rS} d_m$.

Proof. We first show the allocation is valid. For any mobile device within a recurring window Sd_m, it receives one primary burst with size $(r - r_l)Sd'_m$ and S bootstrap bursts with aggregate size $\frac{r-r_l}{r} d_m r_l + d_m r_l (S-1)$. Rearranging the aggregate size of bootstrap bursts leads to:

$$\begin{aligned} \frac{r - r_l}{r} d_m r_l + d_m r_l (S-1) =& d_m r_l - \frac{r_l}{r} d_m r_l + S d_m r_l - d_m r_l \\ =& \frac{rS - r_l}{rS} d_m S r_l = d'_m S r_l. \end{aligned} \tag{5.9}$$

Hence, the number of received bits between adjacent bootstrap bursts is equal to the number of consumed bits before the first primary burst arrives (at reduced-quality with bit rate r_l), and the same balance holds when receiving the primary bursts (at full-quality with bit rate r). Therefore, our allocation is valid.

Second, according to Eq. (5.8), the distance between two consecutive bootstrap bursts is $d'_m + \frac{r_l d_m}{rS} = d'_m + x = d_m$. Hence, our scheme meets the switching delay constraint.

Third, the optimality of the SIMU-S+ scheme can be shown by contradiction using the same idea of the proof in Theorem 10. The details are not given for brevity.

Last, we derive the energy saving achieved by this scheme, in which mobile devices receive both primary and bootstrap bursts all the time. Notice that as we concatenate three bursts together as mentioned above, the total number of bursts in steady stage is $S-1$. In addition, the aggregate receiving time is Srd'_m/R. The γ is then given as $\gamma = 1 - \frac{Srd'_m/R + (S-1)T_o}{Sd'_m}$. □

5.5 Evaluation on a Mobile TV Testbed

In this section, we evaluate our proposed time slicing schemes in a mobile TV testbed. For homogeneous mobile devices, we focus on the SIMU-S scheme because it combines the advantages of the SIMU scheme with the low storage and simple stream management of scalable coding. We also have shown in Section 5.4 that both SIMU and SIMU-S achieve the same energy saving, which is higher than that of SIMU-S+.

5.5.1 Testbed Setup and Metrics

We use the real mobile TV testbed described in Section 3.6 to evaluate the proposed time slicing schemes. We have implemented the SIMU-S time slicing scheme in the broadcast base station. We have also implemented the current time slicing scheme for comparison. To conduct our experiments, we configure the testbed as follows. We use a 5 MHz DVB-H channel with QPSK modulation, which leads to 5.445 Mbps air medium bandwidth according to the DVB-H standard [16]. To evaluate the SIMU-S scheme, we configure the streaming sever to send packets of size 1.5 KB at bit rate $r = 300$ kbps, and we set the reduced-quality bit rate $r_l = 100$ kbps. Therefore, our broadcast network can concurrently broadcast up to 13 TV channels. We set the target maximum channel switching delay to be 500 msec. For each considered time slicing scheme, we broadcast 8 TV channels for 10 minutes. Our Linux server running the base station code could not handle (encapsulate, FEC-encode, etc.) more than 8 TV channels in *real time*. This is why we broadcast only 8 channels.

To conduct statistically meaningful performance analysis, we collect detailed event logs from the base station. The logs contain the start time (in msec) of broadcasting every burst of data and its size. Using these logs, we develop a software utility to emulate the behavior of a large number (1 million) of mobile devices. We generate random channel switching events using Bernoulli trials. For every mobile receiver, we toss a biased coin every second and issue a channel switching command if the trial is success. The new selected channel is randomly chosen from all broadcast channels other than the currently watched one. We set the probability of success in a way that we have on average $w = 100$ sec watch time for each channel. We chose a small w for conservative evaluation, because our proposed schemes consume more energy in the bootstrap stage. In typical cases, the watching time should be longer than 100 sec, and the results will be better than those presented in this section. Figure 5.5 depicts the distribution of the channel switching events with $w = 100$ sec, which shows mobile devices on average change their channels 6 times within the 10-minute broadcast.

We then consider the proposed SIMU-S scheme under extreme conditions by varying the watch time w from 10 to 160 sec. We repeat the above experiment for each w value. We let the watch time $w = 100$ sec. For each time slicing scheme, we broadcast 8 TV channels for 30 minutes and we store the details on individual bursts in log files.

We run the emulator against each log file produced by the actual DVB-H broadcast

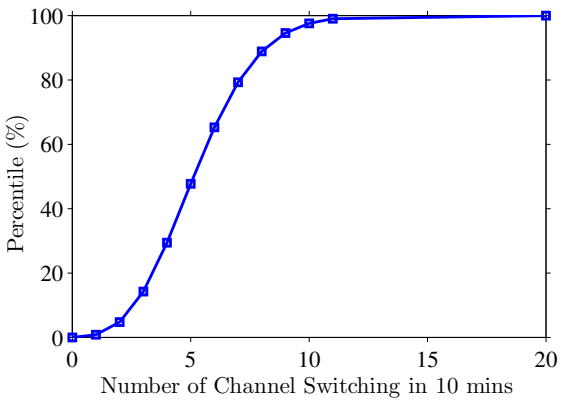


Figure 5.5: Distribution of the emulated channel switching events for 1 million devices.

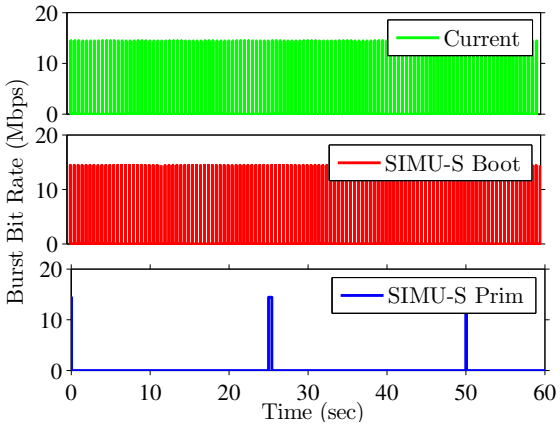


Figure 5.6: Bursts allocated by SIMU-S and the current allocation schemes for a TV channel.

base station and we measure channel switching delays and energy savings. We measure the channel switching delay by searching for the next burst of the new selected TV channel and computing the time difference. We measure the energy saving by computing the fraction of time that the antenna is on. We set the overhead duration $T_o = 100$ msec. When using the proposed schemes, we divide each watching period into a bootstrap stage and a steady stage. We measure the energy savings in both stages, and report the weighted average of them as mobile devices first receive bootstrap bursts and switch to primary bursts whenever the latter ones become available.

5.5.2 Results

Correctness. We first validate the correctness of our testbed implementation. Figure 5.6 plots a sample result of allocated bursts for a TV channel during a minute of broadcast. Burst allocations for longer time periods and for other TV channels are similar. This figure shows that to meet a channel switching delay of 500 msec, the current scheme allocates many short bursts, thus results in low energy savings. While the SIMU-S scheme also employs short bootstrap bursts, it allocates primary bursts with a longer inter-burst distance. By switching from bootstrap to primary bursts, the SIMU-S scheme achieves both short switching delays and high energy savings.

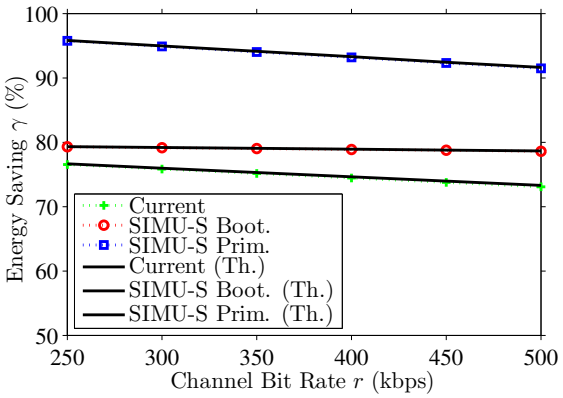


Figure 5.7: Comparison between analytical and empirical energy saving curves.

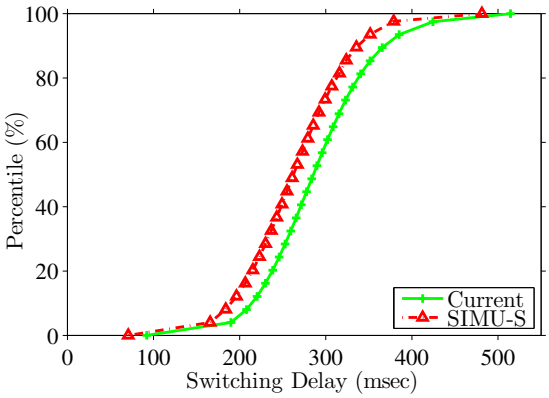


Figure 5.8: Channel switching delay distribution of considered schemes.

Validation of our analytical analysis. We validate the analytical energy saving formulas using the collected empirical data as follows. We consider several TV channel bit rates: from 250 to 500 kbps. For each TV channel bit rate r, we select a reduced-quality bit rate $r_l = 0.2r$ kbps. We then broadcast for 15 minutes at various r using both current and SIMU-S schemes. We measure the energy saving from the collected logs for the SIMU-S and the current time slicing schemes. We also compute the theoretical energy saving from the corresponding equations in Section 5.3 and Section 5.4. We plot the theoretical (denoted by Th.) and the empirical results in Figure 5.7. This figure clearly shows that our analytical formulas closely follow the empirical data collected from the real DVB-H testbed.

Switching delay guarantee. In Section 5.3, we prove that our time slicing schemes provide a guarantee on channel switching delay. Here, we show that our testbed implementation does achieve the guaranteed delays. Figure 5.8 shows the distribution of the channel switching delay observed by the mobile devices switching among randomly chosen channels. This figure shows that the switching delays for all mobile devices are below the target delay of 500 msec.

Energy Saving. Figure 5.9 reports the average energy saving achieved by the current and SIMU-S schemes. This figure shows that while the current scheme results in 74% energy saving, our proposed SIMU-S scheme yields up to 93% energy saving while providing a guarantee on switching delay. Note that, as the current scheme allocates bursts at uniform

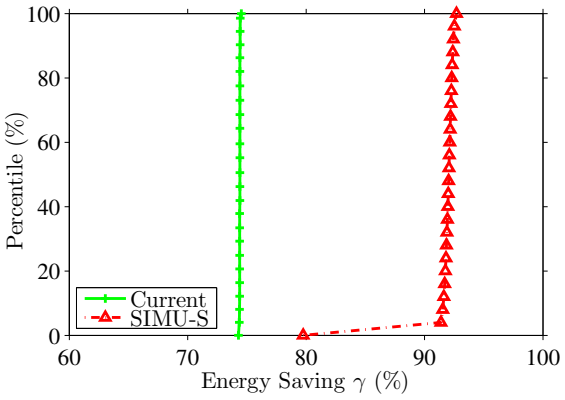


Figure 5.9: Energy saving comparison between SIMU-S and the current schemes.

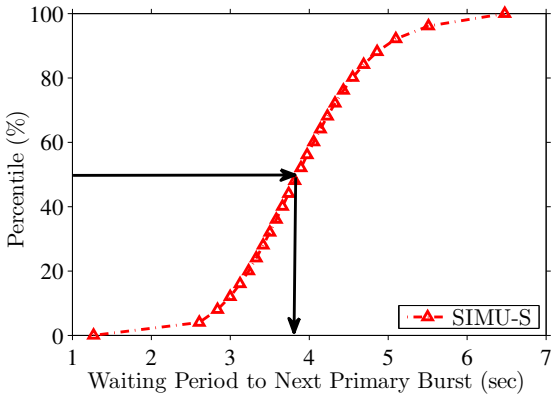


Figure 5.10: Waiting period of time before switching to primary trains.

distance, the average energy saving does not depend on the time of channel switching events. Hence, most mobile devices achieve about 74% energy saving. In contrast, in the SIMU-S scheme, mobile devices achieve slightly diverse energy saving. This is because of the existence of the bootstrap and primary burst trains. Recall that mobile devices first tune for the bootstrap bursts, then switch to the primary bursts once they become available. Figure 5.10 plots the average period of time a mobile device waits in the bootstrap stage until it receives the first primary burst. We observe some diversity in this waiting period, which explains the minor differences in the achieved energy saving for SIMU-S in Figure 5.9. This figure illustrates that it takes mobile devices on average only 3.8 sec to switch to high-quality videos in this test scenario. The short period in the bootstrap stage is desirable, because it means that users will obtain full-quality video sooner, and they may not observe the reduced-quality during the short transition period.

Implication of watch time. We report the energy savings achieved by the SIMU-S scheme under different watch time w. We repeat the experiment several times with various w values: 10, 20, 40, 80, and 160 sec, and we measure the average energy saving achieved by mobile devices. We plot the results in Figure 5.11. This figure shows that the proposed scheme achieves on average 87% even under an extreme condition where users watch each TV channel for only 10 sec. This is in contrast to 74% energy saving achieved by the current scheme, as illustrated in Figure 5.9. As high as 95% energy saving is possible when users

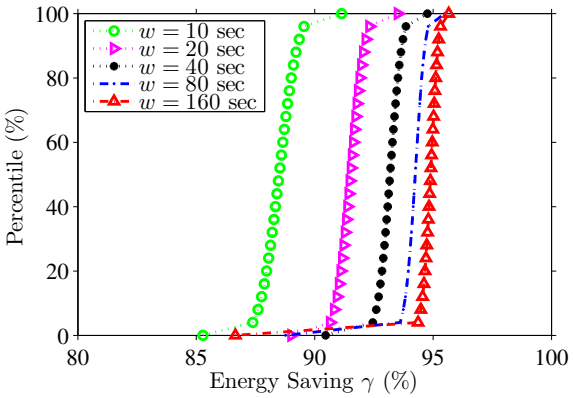


Figure 5.11: Energy saving achieved by SIMU-S scheme under different watch time w.

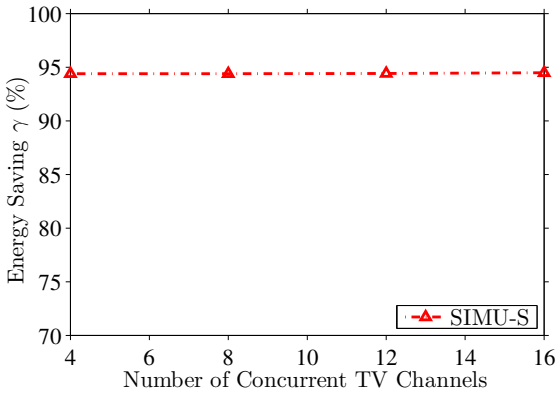
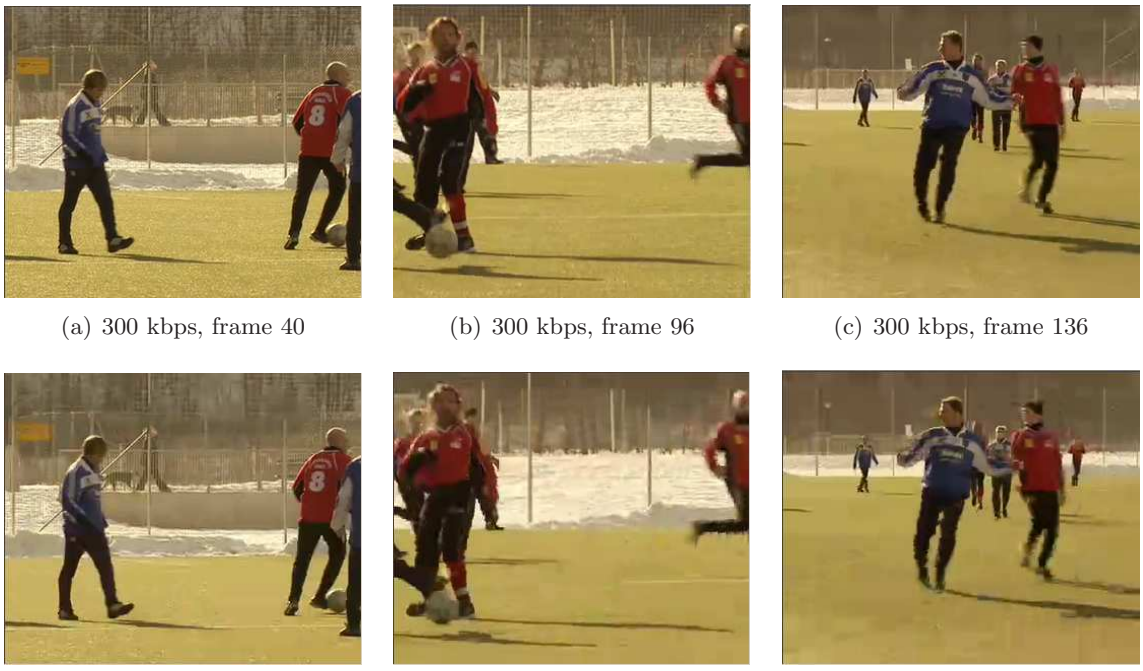


Figure 5.12: Energy saving achieved under different number of TV channels.

continuously watch a TV channel for more than a couple of minutes, which is the common scenario as TV commercials are apart for at least 10 minutes.

Scalability. Since SIMU-S burst allocation scheme realizes the time division multiplexing, each TV channel gets reversed time slots and do not interfere with each other. We use our testbed implementation to validate this. We do this by concurrently broadcasting several TV channels for 15 minutes. We vary the number of TV channels from 4 to 16. Figure 5.12 depicts that number of TV channels does not affect achieved energy saving as the average energy saving remains at more than 94% in all considered number of TV channels.

Video quality. Finally, we study whether the reduced bit rates for bootstrap bursts can provide reasonable video quality. We encode two video sequences, soccer and harbour, using the H.264/SVC reference coder [97]. Both video sequences are in CIF format at 15 frames per second (fps), which is very close to the popular QVGA resolution adopted by many mobile TV devices in the market. We compress these two sequences at various bit rates. To visually inspect the resulting frame quality, we present sample reconstructed frames of each sequence at two bit rates in Figs. 5.13 and 5.14. Notice that several impairments can be observed in the reduced-quality frames, e.g., there are many blocking artifacts around the right-most runner's leg in frame 96 of the soccer sequence when it is encoded at 100 kbps. Nevertheless, the reduced-quality frames provide users rough representations, and enable a much higher energy saving as we have shown above.



(a) 300 kbps, frame 40 (b) 300 kbps, frame 96 (c) 300 kbps, frame 136

(d) 100 kbps, frame 40 (e) 100 kbps, frame 96 (f) 100 kbps, frame 136

Figure 5.13: Sample reconstructed frames from Soccer sequence coded at two bit rates.

(a) 300 kbps, frame 64 (b) 300 kbps, frame 105 (c) 300 kbps, frame 149

(d) 100 kbps, frame 64 (e) 100 kbps, frame 105 (f) 100 kbps, frame 149

Figure 5.14: Sample reconstructed frames from Harbour sequence coded at two bit rates.

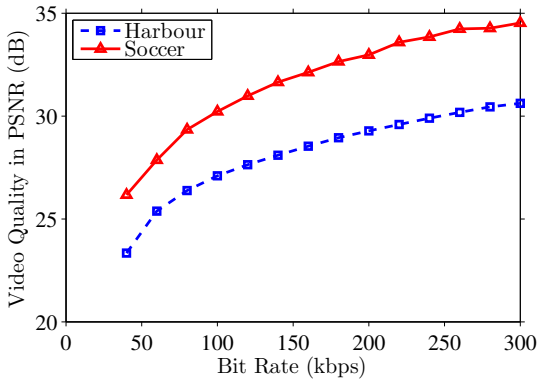


Figure 5.15: Modern video coding standards enable us to transmit reasonable quality videos at bootstrap time.

To better quantify the video quality, we compute the average peak signal-to-noise ratio (PSNR) values of individual coded streams. We plot the rate-distortion (R-D) curves in Figure 5.15. This figure shows that H.264/AVC achieves reasonable video quality even at a low bit rate. For example, 30 dB video quality can be achieved at 300 kbps for the harbour sequence, and at 100 kbps for the soccer sequence. We note that PSNR values above 30 dB are considered fairly good quality [109, pp. 29]. In summary, this experiment shows that video streams coded at a fairly low bit rate could result in acceptable video quality.

5.6 Conclusions

In this chapter, we considered the problem of controlling the channel switching delay in mobile video broadcast networks that use time slicing to save energy. Our objective is to provide a guarantee on the maximum switching delay from a channel to any other channel, without sacrificing energy savings for mobile devices. We analyzed the time slicing scheme used in the current mobile video broadcast networks and showed that it does not minimize the energy consumption for mobile devices. We proposed new time slicing schemes: SIMU, SIMU-S, and SIMU-S+, which are provably optimal in terms of energy saving. Our analysis showed that there are three aspects in mobile video broadcast networks: energy saving, channel switching delay, and bandwidth utilization. These three aspects cannot be optimized at the same time. For example, the current time slicing scheme can achieve full bandwidth utilization, but it trades energy saving for shorter switching delays. Our proposed SIMU and SIMU-S schemes achieve optimal energy savings and small switching delays, but at the expense of reduced bandwidth utilization. Our SIMU-S+ scheme achieves a slightly better energy saving than the current scheme while fully utilizing the bandwidth and meeting a required maximum channel switching delay.

Using our analysis, the appropriate time slicing scheme can be chosen for a given environment: most broadcast networks should adopt SIMU or SIMU-S scheme to achieve the optimal energy saving on mobile devices, and bandwidth saturated broadcast networks should employ SIMU-S+ scheme. We implemented the proposed time slicing schemes in a mobile TV testbed. We conducted real-time broadcast experiments and emulated a large number of mobile devices that are randomly switching among many TV channels. Our experiments confirmed the correctness of our analysis and demonstrated that using our time slicing schemes, an energy saving as high as 95% can be achieved while guaranteeing a

channel switching delay of at most 500 msec. In contrast, the current time slicing scheme can achieve 500 msec switching delay, but at a much lower energy saving of 73%.

Chapter 6

Supporting Heterogeneous Receivers

In this chapter, we study the problem of supporting heterogeneous devices in mobile video broadcast networks. We first show that existing base stations cannot efficiently support mobile devices with heterogeneous resources. We then propose two time slicing schemes to broadcast scalable video streams. We implement and evaluate our solutions in a real mobile TV testbed.

6.1 Introduction

Efficiently supporting heterogeneous devices, as described in Problem 4, is important because modern mobile devices have diverse resources such as screen resolution, decoder capability, and battery capacity. Therefore, encoding each video into a single video stream may be inefficient to some mobile devices, and may deny some mobile devices from the mobile video broadcast service. Network operators can partially cope with the problem by broadcasting each video in multiple versions. This, however, results in bandwidth inefficiency. This inefficiency becomes even more severe if we categorize mobile devices into groups by not only device models but also by working conditions of the same device. For example, mobile devices with low battery levels or in poor wireless channel conditions may prefer to receive at a lower bit rate to save energy and/or reduce bit error rate.

Instead of multi-version broadcasting, we propose to use scalable video streams to support heterogeneous mobile devices. Scalable video coding is described in Section 2.2.2. We use scalable video coders to encode each video into a single stream with multiple layers, and broadcast each layer only once. Each mobile device then chooses and renders the substream that is most appropriate to its capability and condition. Scalable video broadcasting, however, is quite challenging for the base station broadcasting multiple video streams. This is because the base station must prepare bursts for all concurrently broadcast video streams to ensure that: (i) mobile devices can receive enough data for smooth playouts, and (ii) no bursts intersect with each other in time. Furthermore, preparing bursts for scalable videos is complicated because the dependency among layers must be carefully considered.

Our problem is to encapsulate and broadcast video streams encoded in a scalable manner, so that heterogeneous mobile devices can render the most appropriate video substreams to achieve high energy saving and low channel switching delay. The appropriate streams depend on the device capability and the target energy consumption level. More precisely, we first analyze the current mobile video broadcast networks and we show that they are not efficient for scalable coded streams. This is done by first showing that limitations of the current systems, and then analytically show that these broadcast schemes lead to lower energy saving compared to our proposed schemes. We propose two new broadcast schemes for scalable video streams with arbitrary layer bit rates, and we analytically analyze their performance. The two proposed schemes are flexible in terms of bit rates of individual layers, which allow the coded video streams to be better matched with the capability of mobile devices. We analytically prove that these schemes achieve high energy saving and result in low channel switching delays. We implement the broadcast schemes in a real mobile TV testbed to show their practicality and efficiency. The experimental results confirm our analytical analysis, and indicate that the proposed schemes allow mobile devices to trade perceived quality for energy saving, as they can opt to receive a smaller substream to prolong battery lifetime. Furthermore, very short channel switching delay, as low as a few hundred milliseconds, can be achieved using one of the proposed schemes.

6.2 Related Work

The energy saving of mobile devices has been studied for mobile video broadcast networks, where video streams are coded in nonscalable fashion. See Section 3.2 for a survey. In

addition, reducing channel switching delay of mobile video broadcast networks has also been explored in the literature. See Section 5.2 for a survey.

Scalable video coding has been adopted in mobile TV networks to support mobile devices under diverse reception conditions [110]. For example, unequal error protection (UEP) methods were proposed to improve video quality for mobile devices with bad radio receptions [111,112]. Haddadi et al. [111] propose to transmit the base layer and the enhancement layers with different modulation and coding schemes, so that the base layer is sent over a more robust low bit rate channel while the enhancement layer is sent over a less robust high bit rate channel. This is called hierarchical modulation and channel coding, and is supported by many broadcast networks including DVB-T [113]. Hellge et al. [112] consider the layering dependency in multi-layer video streams, and propose to use parity bits of higher layers to protect lower layers, which need to be more resilient to errors as they are more critical to successful decoding. None of these works considers burst transmission for saving energy, thus they are orthogonal to our work.

Multicast of scalable video streams over the Internet has been studied in the literature and many protocols and algorithms have been proposed to support multicast routing, resource reservation, robustness, and flow and congestion controls [114, 115]. None of these proposals is applicable in mobile TV networks, because they are single-hop broadcast networks, rather than the multi-hop Internet. For example, in RLM (Receiver-driven Layered Multicast) [116], different layers of a video stream are sent to different multicast groups, and receivers periodically join the next higher layer's group until experiencing excessive packet loss. RLM is not useful in mobile TV networks because of their broadcast nature: *more* receivers do not incur *higher* network loads, and packet loss ratio on a mobile device is independent of how much data it receives. Hence, the previous works on multicast over the Internet or wireless networks are not applicable in mobile TV networks.

6.3 Problem Statement

In this section, we first define the problem considered in this chapter. We then describe the operation of current mobile video broadcast networks. We list all symbols used in the chapter in Table 6.1 for quick reference.

Table 6.1: List of symbols used in supporting heterogeneous devices.

Sym	Definition	Sym	Definition
S	no. channels	γ	energy saving
R	burst bit rate	γ_c	γ of class c in steady stage
C	no. layers	γ_b	γ in bootstrap stage
r	channel bit rate	b	base layer burst size
r_c	bit rate of layer c	d	channel switching delay
T_o	overhead duration	d_m	maximum switching delay

6.3.1 Burst Transmission to Heterogeneous Mobile Devices

We consider a mobile video broadcast network in which a base station concurrently broadcasts multiple TV channels over a shared air medium with bandwidth R kbps to mobile devices with *heterogeneous* resources. The network bandwidth is divided among TV channels. Each TV channel carries a video stream and is assigned a bit rate of r kbps. To support heterogeneous mobile devices, each video stream is encoded into C layers using scalable video coders, where each layer c ($1 \leq c \leq C$) has a bit rate of r_c kbps. The base station broadcasts each video stream as a series of bursts, and a mobile device receives a burst of data and turns off its receiving circuit until the next burst of the same video stream to save energy. This is called time slicing.

Time slicing is critical to the quality of service in mobile video broadcast networks for two reasons. First, time slicing enables mobile devices to save energy by turning off their receiving circuits while not receiving bursts. Higher energy saving results in longer battery lifetime and watch time. Second, time slicing has an impact on channel switching delay, which refers to the time a user waits before s/he starts viewing a selected channel when a change of channel is requested by that user [29,33]. High and varying channel switching delay degrades view experience because many users quickly flip through numerous TV channels before they decide to watch specific ones.

Next, we formally define the energy saving and the channel switching delay. The energy saved by a mobile device because of time slicing is denoted by γ, and it is calculated as the ratio of time the receiving circuit is in off mode to the total time [16, 19]. When computing the energy saving, we need to consider the wake-up time of the receiving circuits on mobile devices to receive the next burst. This is because it takes receiving circuits some time to power up and lock onto the radio signals before data can be demodulated [16]. This period is called overhead duration and is denoted as T_o. T_o is not negligible, and can be as high

as 250 msec [9, 10, 16]. Furthermore, in Section 2.6, we empirically show that a recent Nokia cellular phone has T_o in the range of 80–140 msec. The channel switching delay d consists of several components, in which the frame refresh delay and time slicing delay are the two dominating contributors [29, 33]. As mentioned in Section 1.2.3, we only consider time slicing delay, and assume all other components of the channel switching delay are fixed. Furthermore, we define d_m as the maximal (the worse case) channel switching delay.

Using the above notations, we can restate Problem 4 as follows. Consider a mobile video broadcast network with air medium bandwidth R kbps shared among S TV channels, where each TV channel has a bit rate of r kbps. Every TV channel is encoded into C layers, and layer c ($1 \leq c \leq C$) has a bit rate of r_c kbps, where $\sum_{c=1}^{C} r_c = r$. Mobile devices are classified into C classes so that devices in class c receive and decode all layers $\bar{c}$, where $\bar{c} \leq c$. The coded streams are encapsulated into bursts, and broadcast over the shared air medium. Design a broadcast scheme such that mobile devices in any class c achieves high energy saving and low channel switching delay from any channel to any other channel. A broadcast scheme assigns IP packets to individual bursts, and specifies the start time of each burst. Solving this problem at a base station allows users to watch more video streams (due to longer battery lifetime), while keeping channel switching delay short. This will improve user satisfactory and eventually increase the number of subscribers as well as profits of network operators.

6.3.2 Burst Preparation in Mobile TV Networks

We present how mobile video broadcast base stations prepare transmission bursts in the following. We use DVB-H network as an example. In a DVB-H network, each mobile TV base station consists of three main components: video server, IP encapsulator, and modulator, as illustrated in Figure 2.1. The video server encapsulates video data, pre-encoded or live, into RTP packets, and sends these packets over an IP network to the IP encapsulator. The IP encapsulator receives these IP packets and puts them in MPE (multiprotocol encapsulation) frames. MPE frames are used by IP encapsulator for preparing transmission bursts of a specific TV channel, and each MPE frame is sent as *one* burst. The MPE frame can optionally be protected by Reed-Solomon (R-S) codes. This is achieved by extending the MPE frame into the MPE-FEC frame, which consists of FEC parity bytes. Since mobile devices are vulnerable to bad radio channel conditions, MPE-FEC frames are important as they provide better error resilience. We note that the name IP encapsulator is a bit

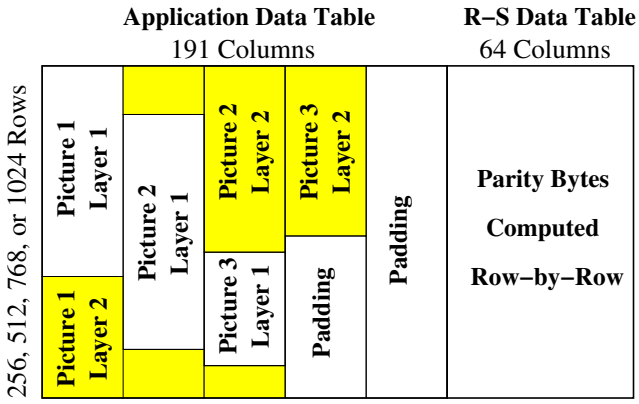


Figure 6.1: The structure of MPE-FEC frame. IP packets are sequentially placed in the Application Data Table.

misleading as it actually encapsulates IP packets into bursts, thus it should be better called *burst encapsulator*. We, however, use the term IP encapsulator in this chapter, following previous works in the standardization documents [16] and in the literature [19, 72].

Figure 6.1 reveals the structure of an MPE-FEC frame, which is divided into two parts: an application data table (ADT) that carries IP packets and an R-S data table (RDT) that carries the parity bytes. To compute the parity bytes, received IP packets are *sequentially* placed in the ADT column-by-column, from left to right. Zeros are padded in the remaining space of the ADT if there are not enough data to fill the ADT. Once the ADT is full (by data and/or zeros), the parity bytes are computed row-by-row, and stored in the RDT. After the parity bytes are computed, the whole MPE-FEC frame is sent, column-by-column, as a burst. Note that, the padded zeros are for computing parity bytes only; they are not transmitted over the wireless medium [9]. We study the problem of designing broadcast schemes to optimize the quality of service, therefore the proposed schemes will be implemented in the IP encapsulator.

Despite different terminology, MediaFLO base stations also have to prepare transmission bursts in a time-slicing manner [117]. More specifically, MediaFLO base stations transmit signals in a superframe structure, where each superframe consists of four frames and each frame has 250 OFDM symbols. A TV channel is assigned several OFDM symbols in every frame, and this assignment defines a period of time that mobile devices *must* turn on their receiving circuits to receive data. Therefore, MediaFLO base stations prepare transmission bursts in the form of assigning OFDM symbols to individual TV channels, and most of our

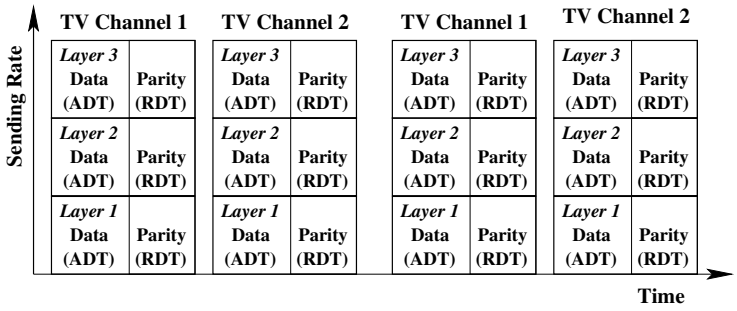


Figure 6.2: Parallel Services.

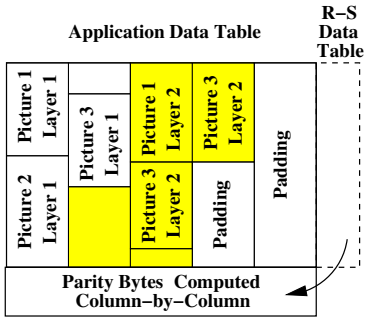


Figure 6.3: Layer-Aware FEC frame.

works in this chapter are also applicable to MediaFLO networks.

6.4 Broadcasting Scalable Streams in Current Systems

Traditional, nonscalable, coded streams must be decoded in their entirety, and are not efficient for mobile devices with heterogeneous resources. This is because all mobile devices have to receive the complete video streams even though some of them do not have enough resources to render the complete streams. In contrast, scalable video coders encode each TV channel into a single video stream that can be sent and decoded at various bit rates. This is achieved by extracting substreams from the complete (original) stream, where each substream can be decoded and displayed at a lower perceived quality.

While scalable coded streams enable efficient substream extractions, broadcasting them in mobile video broadcast networks is challenging because of the dependency among layers of the same coded stream. In this section, we present three schemes to broadcast scalable video streams in current systems, and we show their shortcomings.

6.4.1 Single Service: SS

To support heterogeneous mobile devices, network operators can upgrade the video server (see Figure 2.1) to support scalable streams, and keep other components of the network unchanged. That is, each scalable stream is transmitted as a TV channel, and we refer to this scheme as single service (SS). With the SS scheme, existing broadcast networks can be

used to broadcast scalable video streams. Unfortunately, this "patched" base station is not efficient as we show in the following illustrative example.

We consider a small time window of three pictures[1], where each picture is coded into two layers. We assume that each layer is encapsulated into a single IP packet, and these packets are sent by the video server and received by the IP encapsulator in the following order: (picture 1, layer 1), (picture 1, layer 2), (picture 2, layer 1), (picture 2, layer 2), (picture 3, layer 1), (picture 3, layer 2). Since the IP encapsulator *sequentially* places IP packets in the receiving order within the ADT part of MPE-FEC frames, these packets are stored into a frame as illustrated in Figure 6.1. This MPE-FEC frame is then sent over the broadcast network as a burst. Consider a mobile device that can only render the base layer (layer 1), this mobile device *must* receive and process the complete burst for two reasons. First, IP packets belonging to the base layer (unshaded in the figure) are scattered all over the MPE-FEC frame, and a deep inspection (at RTP or video-coding layer) is required to locate them. Second, the parity bytes are computed over IP packets from various layers, and are useless if some IP packets are not received. Receiving complete bursts degrades the energy saving of that mobile device, because it has to open the receiving circuit for a longer time period to receive some data that will be *dropped* eventually.

6.4.2 Parallel Services: PS

The example in Section 6.4.1 shows that single service leads to *no* additional energy saving for mobile devices that cannot render the complete video streams. To cope with this issue, we need to design a better IP encapsulator that takes the layering structure of scalable streams into considerations when preparing transmission bursts. One way to achieve this is to send each layer of a TV channel as a parallel service (PS), which can be implemented using multiple IP streams with different multicast IP addresses, or using multiple parallel elementary streams [16, Section 8.6]. Figure 6.2 shows an example of broadcasting two TV channels with three layers, where each block inside bursts is a parallel service and carries IP packets of a specific layer only. Compared to single service, parallel service scheme supports efficient demultiplexing of IP packets to individual layers based on either IP addresses or MPEG-2 PIDs (packet IDs). This allows mobile devices to extract substreams without inspecting the complete stream, and thus reduces its processing overhead. Unfortunately,

[1] We interchangeably use frame and picture throughout this chapter.

as all services are sent in parallel, mobile devices have to open their receiving circuits for the complete burst duration. Therefore, all mobile devices achieve the same energy saving despite how many layers they receive and decode. Observe that parallel service is inefficient because mobile devices may receive and discard data (in irrelevant layers) that are *useless* to them. This leads to longer on time of receiving circuits, and thus lower energy saving.

6.4.3 Layer-Aware FEC: LAF

To cope with the inefficiency of parallel service, the IP encapsulator may rearrange the IP packets received from streaming services, so that packets belonging to layer c are sent before packets belonging to layer $c+1$. This allows mobile devices that do not decode the complete scalable streams to turn off their receiving circuits before each burst ends. If we reuse the illustrative example given in Section 6.4.1, the IP packets should be sent in the following order: (picture 1, layer 1), (picture 2, layer 1), (picture 3, layer 1), (picture 1, layer 2), (picture 2, layer 2), (picture 3, layer 2), as illustrated in Figure 6.3. The layer number can be prepended before the MPE header as an one-byte extension header, which allows mobile devices to efficiently determine the boundaries between layers, e.g., between (picture 3, layer 1) and (picture 1, layer 2) in this example,

However, even after reordering IP packets, mobile devices still have to receive complete bursts in order to perform error correction, which again prevents them from getting higher energy saving. This is because the FEC parity bytes are sent after all the IP packets, at the end of each burst. To address this issue, we can compute parity bytes *column-by-column*, and send these bytes right after each column of data bytes. This allows mobile devices to perform error corrections without receiving complete bursts. That is, mobile devices can receive partial bursts and turn off the receiving circuits to save energy. We call this new frame format as Layer-Aware FEC (LAF) frame, which is illustrated in Figure 6.3.

While LAF frame allows mobile devices to efficiently receive and extract substreams, it has several drawbacks. First, LAF does not comply to mobile TV standards, which causes compatibility issues between the base station and mobile devices. Second, implementing LAF requires significant changes as error corrections are usually done in hardware/firmware for the sake of performance. Third, computing parity bytes column-by-column makes the FEC decoder vulnerable to bursty channel errors because it does not provide virtual time interleaving as by MPE-FEC frames [9]. Most importantly, in Section 6.5, we prove in

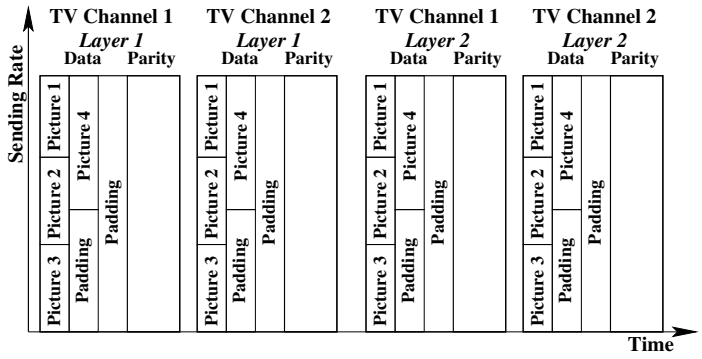


Figure 6.4: Proposed broadcast Scheme: GLATS.

Lemma 3 that the LAF scheme achieves lower energy saving compared to the scheme developed in that section.

6.4.4 Limitations of Current Systems

We have presented three schemes to broadcast scalable video streams in current systems: single service (SS), parallel service (PS), and layer-aware FEC (LAF) frame. The SS and PS schemes require all mobile devices, despite which classes they are in, to receive complete scalable streams. Therefore, they result in *no* energy saving differentiation as all mobile devices turn on their receiving circuits for the same amount of time. While the LAF scheme enables energy saving differentiation, it is: (i) not standard compliant, (ii) hard to implement, (iii) vulnerable to bad channel conditions, and (iv) achieves lower energy saving than our broadcast scheme proposed in Section 6.5. Hence, we conclude that current mobile TV broadcast networks cannot efficiently support scalable video streams, and we need to design better broadcast schemes to solve Problem 4.

6.5 Generalized Layer-Aware Time Slicing

We propose a new broadcast scheme which we call Generalized Layer-Aware Time Slicing (GLATS). We analytically analyze the performance of the proposed scheme, and we conduct numerical analysis on it using typical network parameters.

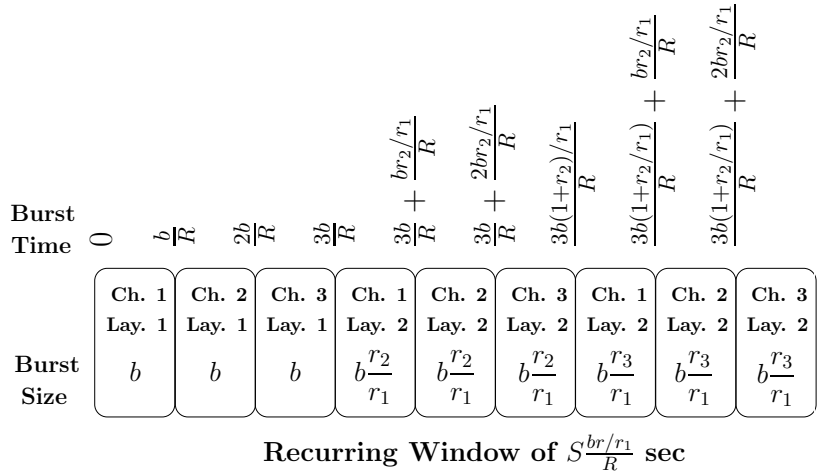


Figure 6.5: An illustrative example of the proposed GLATS scheme.

6.5.1 Overview

The GLATS scheme works on a recurring window, and its key feature is that the IP encapsulator prepares a different burst (or MPE-FEC frame) in the recurring window for each layer of every TV channel. More specifically, every burst in the GLATS scheme consists of IP packets from the same layer of the same TV channel, which allows mobile devices to safely skip bursts that contain IP packets for irrelevant layers and save more energy. To illustrate, Figure 6.4 shows an example of the GLATS scheme, in which all IP packets in the left-most burst belong to the base layer of TV channel 1, while IP packets in the third burst belong to layer 2 of TV channel 1. With the GLATS scheme, mobile devices that are rendering TV channel 1 at the base layer quality do *not* need to receive the third burst, because it is of no use to them. More importantly, mobile devices know which layer the IP packets are in, even before receiving a burst. Therefore, mobile devices need *not* open their receiving circuits and inspect IP packets for substream extractions, and thus more energy can be saved. In fact, no additional signaling from the base station to mobile devices is required: to determine which bursts (layers) to receive, mobile devices only need to know the total number of layers C, which is already sent to them for decoding the scalable stream. Furthermore, the GLATS scheme naturally works with existing MPE-FEC frame, because whenever a mobile device decides to decode layer c, it has to receive all IP packets in layer c for successful video reconstruction. Therefore, all IP packets in ADT will be received before error correction, and the FEC decoder (implemented in hardware/firmware) can work as-is.

We derive the GLATS scheme in the following, and give the burst start time and the

burst size of each layer of individual TV channels. First, the number of TV channels that can be concurrently broadcast is: $S = \lfloor R/r \rfloor$. We let b kb be the burst size of base layers, which is a system parameter. The burst size for layer c is proportionally set to br_c/r_1 kb. Since the recurring window consists of a burst for every layer of each TV channel, the aggregate burst size of the complete stream of a TV channel within the window is $\sum_{c=1}^{C} br_c/r_1 = b\sum_{c=1}^{C} r_c/r_1 = br/r_1$ kb. Since the broadcast bandwidth is R kbps, the recurring window size is $\left(\frac{br/r_1}{R}\right) \times S = (brS)/(r_1R)$, where $\frac{br/r_1}{R}$ is the amount of time the GLATS scheme *must* reserve for a TV channel, and there are S TV channels in total. Finally, the GLATS scheme produces the time slicing schedule with bursts

$$\textbf{GLATS}: \left\{ \left\langle s,\ c,\ \frac{b\sum_{i=1}^{c-1} r_i/r_1}{R} S + \frac{br_c/r_1}{R}(s-1),\ br_c/r_1 \right\rangle \Big| \right.$$
$$\left. \forall s = 1, 2, \ldots, S,\ \ c = 1, 2, \ldots, C \right\}, \tag{6.1}$$

where the third element of the 4-tuple is the burst start time, and the last element is the burst size.

Figure 6.5 gives a simple example of the GLATS scheme with three TV channels, where each TV channel is scalably coded into three layers. Notice that the GLATS scheme aligns layers c of all S TV channels together, which occupy the air medium for time $\frac{br_c/r_1}{R} \times S$. Therefore, the start time of the burst of TV channel 1 layer c is given by $\frac{b\sum_{i=1}^{c-1} r_i/r_1}{R} S$, which leads to the start time (the third element of the 4-tuple) in Eq. (6.1).

We mention that the GLATS scheme is fairly general, as it supports layers with *diverse* bit rates. In contrast, the Layer-Aware Time Slicing (LATS) scheme proposed in our earlier work [36] assumes all layers have the same bit rate, i.e., $r_c = r/C$ for all $c = 1, 2, \ldots, C$. Hence, the LATS scheme is a special case of the newly proposed GLATS scheme.

6.5.2 Analytical Analysis

In the next theorem, we formally prove the correctness of the proposed GLATS scheme, and we quantify its performance in terms of energy saving and channel switching delay.

Theorem 12. *The GLATS scheme (Eq. (6.1)) specifies a feasible time slicing scheme for a recurring windows of $\frac{brS}{r_1R}$ sec, where (i) no two bursts overlap with each other, and (ii)*

bursts are long enough to send data for all mobile devices to playout until the next burst. Furthermore, the energy saving achieved by mobile devices in class c is given by:

$$\gamma_c = 1 - \frac{\sum_{i=1}^{c} r_i}{rS} - \frac{RT_o c r_1}{brS} \quad \textit{where } c = 1, 2, \ldots, C. \tag{6.2}$$

Finally, the maximum channel switching delay is:

$$d_m = b/r_1. \tag{6.3}$$

Proof. First, sending a burst of br_c/r_1 kb takes time $\frac{br_c/r_1}{R}$ sec to transmit. Thus. by the definition of Eq. (6.1), the resulting scheme has no overlapping bursts. Second, consider any arbitrary layer c, where $1 \leq c \leq C$, the required amount of data for smooth playout is:

$$\frac{brS}{r_1 R} \times r_c \leq \frac{br}{r_1 R} \times \frac{R}{r} \times r_c = b\frac{r_c}{r_1},$$

where the inequality comes from the definition of S. This inequality shows that the scheduled time period is long enough to carry the playout data.

For energy saving, observe that mobile devices in class c turn on their receiving circuits for c times in each recurring window. Combining this with the aggregate burst size, we have:

$$\gamma_c = 1 - \frac{\frac{b\sum_{i=1}^{c} r_i/r_1}{R} + cT_o}{\frac{brS}{r_1 R}}.$$

In this equation, the first term of the numerator accounts for the time to receive actual video data, the second term of the numerator represents the total overhead durations, and the denominator is the recurring window size. Manipulating this equation yields Eq. (6.2).

For channel switching delay, we first consider mobile devices in class 1. The worst case happens when a user switches to a channel s right after the burst for layer 1 of channel s is broadcast: the user has to wait for the complete recurring window, i.e., $d_m = \frac{brS}{r_1 R}$. Following the definition of S, we write:

$$d_m = \frac{brS}{r_1 R} \leq \frac{br}{r_1 R} \times \frac{R}{r} = b/r_1,$$

which yields Eq. (6.3). This result can be extended to any class c: mobile devices, despite how many layers they receive, can start playing out with the base layer (layer 1) whenever it arrives. Mobile devices gradually add enhancement layers whenever they are available for incremental quality improvements. □

This theorem shows that GLATS scheme is correct and allows mobile devices in different classes to receive and render at different perceived quality, while achieving proportional energy saving. It also shows that the maximal channel switching delay is the same for mobile devices in all classes. This is a nice property, as the GLATS scheme offers *consistent* guarantees on the channel switching delay.

In the next lemma, we compare the performance of the GLATS scheme against and the LAF scheme proposed in Section 6.4.3.

Lemma 3. *The GLATS scheme achieves higher energy saving than the LAF scheme for class c mobile devices if $c \neq C$. These two schemes lead to the same energy saving for class C mobile devices.*

Proof. With the LAF scheme, mobile devices in class c ($c = 1, 2, \ldots, C$) receive a fraction/prefix of every burst, and turn off their receiving circuits earlier to save energy. Since b is the base layer bit rate in the GLATS scheme, we consider a LAF scheme, where each TV channel has C bursts and their aggregate size is br/r_1, for a fair comparison. Observe that class c mobile devices must open their receiving circuits for $\frac{b\sum_{i=1}^{c} r_i/r_1}{R}$, since the network bandwidth is R. Moreover, all mobile devices must turn on their receiving circuits C times in every recurring window of $(brS)\Big/(r_1 R)$. Therefore, we write the energy saving achieved by class c mobile devices as:

$$\begin{aligned}\hat{\gamma}_c &= 1 - \frac{\frac{b\sum_{i=1}^{c} r_i/r_1}{R} + CT_o}{\frac{brS}{r_1 R}} \\ &= 1 - \frac{\sum_{i=1}^{c} r_i}{rS} - \frac{RT_o C r_1}{brS} \quad \text{where } c = 1, 2, \ldots, C. \end{aligned} \tag{6.4}$$

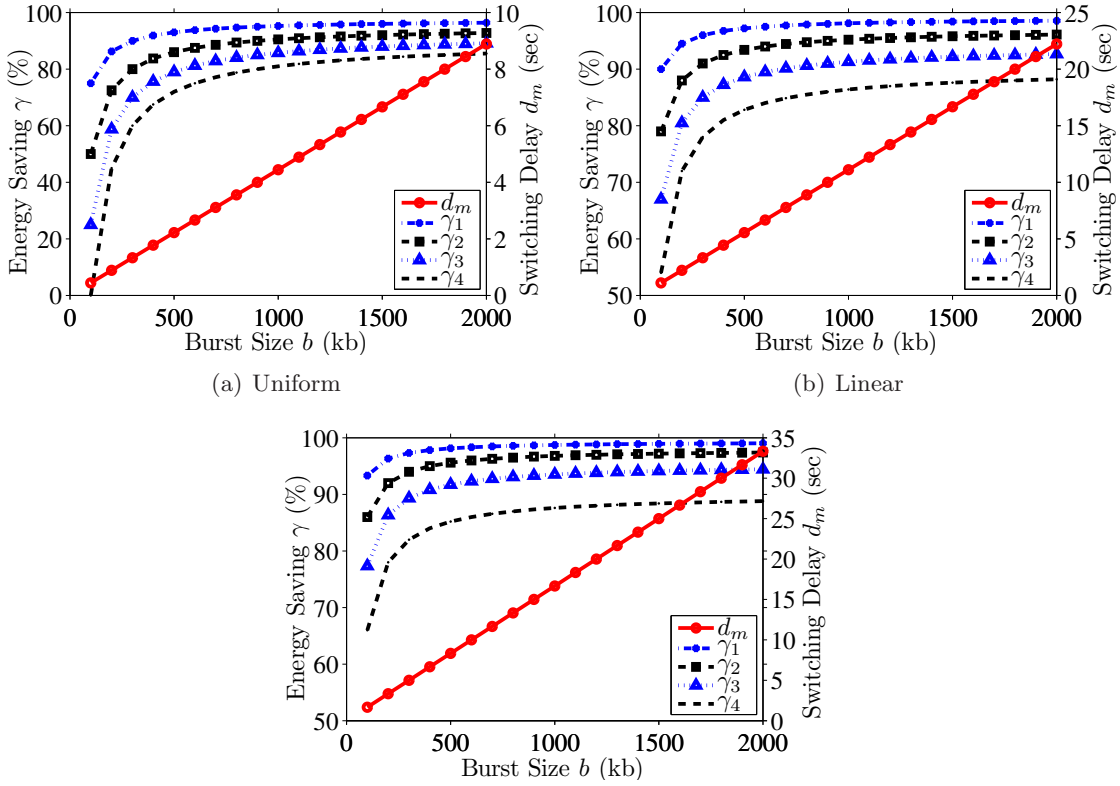
Comparing Eq. (6.4) against Eq. (6.2) yields the lemma. □

This lemma shows that the GLATS scheme outperforms the LAF scheme in terms of energy saving, as aforementioned in Section 6.4.3.

The following corollary is a direct result of Theorem 12.

Corollary 2. *Consider a special case of the GLATS scheme, where all layers have the same bit rate, i.e., $r_c = r/C$ for all $c = 1, 2, \ldots, C$. The energy saving achieved by class c mobile devices is given as:*

$$\gamma_c = 1 - \frac{c}{CS} - \frac{RT_o c}{bCS}.$$



(a) Uniform

(b) Linear

(c) Exponential

Figure 6.6: Diverse energy saving supported by the GLATS scheme, and resulting channel switching delay.

Moreover, the maximal channel switching delay is given as:

$$d_m = bC/r.$$

This corollary considers simplified scalable streams where layers have uniform bit rates, and quantifies the performance of the GLATS scheme under such assumption. We notice that we call this simplified scheme as Layer-Aware Time Slicing (LATS) scheme in our previous results [36], and this corollary is consistent with our earlier analytical analysis presented in [36].

6.5.3 Numerical Analysis and Discussion

We apply typical network parameters to numerically study the performance of the GLATS scheme. More precisely, we consider a mobile video broadcast network with bandwidth

$R = 9$ Mbps, and several TV channels that are encoded into scalable streams at full rate of $r = 900$ kbps. Each of these scalable stream is divided into four layers. We note that the GLATS scheme supports layers with diverse bit rates, and we consider three different scenarios to assign bit rates to individual layers, which are: (i) uniform, in which $r_1 = r_2 = \cdots = r_C = r/C$, (ii) linear, in which $r_c = c \times \frac{2r}{(1+C)C}$, and (iii) exponential, in which $r_c = 2^{c-1} \times \frac{r}{2^C-1}$. We then assume the overhead duration $T_o = 100$ msec, and we vary the base layer burst size b. We compute the energy saving and channel switching delay of mobile devices in different classes using formulas derived in Theorem 12.

We plot the results in Figure 6.6, which clearly shows that the GLATS scheme allows mobile devices to achieve a wide range of energy saving values. For example, letting $b = 1000$ kb, 75% to 95% energy saving is possible in uniform scenario, and 85% to 95% energy saving can be achieved in the other two considered scenarios. The range of supported energy saving is even larger when b is smaller. This energy saving diversity enables mobile devices that are short of resources to be conservative on energy for longer watch time, while others can render TV channels at higher perceived quality. We mention that this diversity may come at an expensive of high channel switching delay. For example, as illustrated in Figure 6.6(a), the channel switching delay is about 5 sec when $b = 1000$ kbps, which may not be desirable for some users. A simple way to cope with long delays is to send bursts more often, which is equivalent to reducing b. Smaller b values, however, may lead to low energy saving as illustrated by Figure 6.6. For example, in Figure 6.6(a), setting $b = 200$ kb results in about 40% energy saving for mobile devices that render the complete video streams, which is significantly smaller than the 85% energy saving that is achieved when $b = 2000$ kb. Therefore, reducing b is not an efficient way to control channel switching delays, and we need to develop a better solution.

6.6 Generalized Layer-Aware Time Slicing with Delay Bound

We propose a new broadcast scheme to achieve energy saving diversity without incurring long channel switching delays, and we refer to it as Generalized Layer-Aware Time Slicing with Delay Bound (GLATSB). We analytically study its performance, and we numerically analyze it using common network parameters.

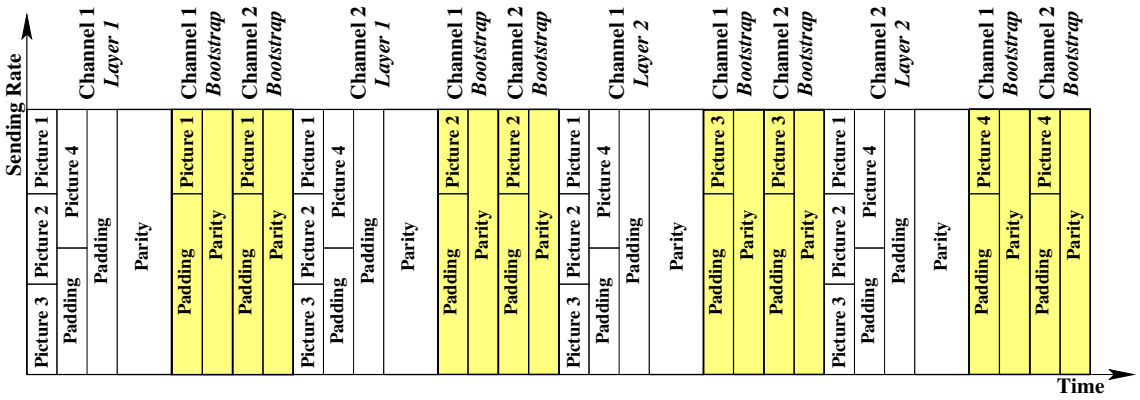


Figure 6.7: The proposed broadcast scheme with switching delay bound: GLATSB.

6.6.1 Overview

The GLATSB scheme is an extension of the GLATS scheme presented in Section 6.5, and it aims to reduce channel switching delays. The delay reduction is based on the following observation. Long channel switching delays are partially due to the dependency among different layers. This is because, despite how many layers a mobile device plans to receive, it cannot decode the coded stream until the base layer (layer 1) arrives, and the waiting time can be as long as the whole recurring window. To better control the channel switching delay, we propose to insert short and frequent *bootstrap* bursts between any two adjacent bursts defined in the GLATS scheme, and send the base layers of TV channels using these bootstrap bursts. We refer to the bursts defined in the GLATS scheme as *normal* bursts.

Figure 6.7 illustrates how we insert the bootstrap bursts: a small piece of base layer from each TV channel is sent between any two normal bursts. This figure shows that two bootstrap bursts for TV channel 1 and 2, respectively, are added between any two normal bursts. Since the bootstrap bursts are sent very often, the user who switches to a new channel can receive the bootstrap bursts and start playing the base layer very quickly. Upon reaching the next normal burst for layer 1 of the selected TV channel, mobile devices switch over to normal bursts. Normal bursts provide higher energy saving, because they are longer, and thus the overhead duration T_o is relatively insignificant to them.

We derive the GLATSB scheme in the following, and give the burst start and the burst size of each layer of individual TV channels. First, the number of TV channels that can be concurrently broadcast is: $S = \lfloor (R)/(r+r_1) \rfloor$. This is because we send each base layer *twice*

in the recurring window: both in bootstrap and normal bursts. We let b be the normal burst size for base layers, and br_c/r_1 be the normal burst size for layer c, where $c = 1, 2, \ldots, C$. b is a system parameter. Then, the recurring window size is: $\Big(b(r_1+r)S\Big)/(r_1 R)$ sec. Compared to that of the GLATS scheme, the window size of the GLATSB scheme has an additional r_1 in the numerator to accommodate the inserted bootstrap bursts, which essentially carry base layers that has a bit rate of r_1 kbps. Next, we define $\bar{b}(c) = br_c/r$ be the aggregate burst size of all bootstrap bursts of a TV channel s ($s = 1, 2, \ldots, S$) between its layer c and $c+1$ normal bursts. Since there are S normal bursts between any two consecutive normal bursts belonging to the same TV channel, the size of a bootstrap burst after a layer c normal burst is $\bar{b}(c)/S = (br_c)/(rS)$. Finally the GLATSB scheme produces the time slicing schedule with bursts

$$\textbf{GLATSB (normal)}:$$
$$\Bigg\{\Big\langle s,\ c,\ b\frac{\sum_{i=1}^{c-1} r_i/r_1}{R}S + \frac{\sum_{i=1}^{c-1}\bar{b}(i)}{R}S + \frac{br_c/r_1}{R}(s-1)+$$
$$\frac{\bar{b}(c)(s-1)}{R},\ \frac{br_c}{r_1}\Big\rangle \ \Big|\ \forall s = 1, 2, \ldots, S,\ c = 1, 2, \ldots, C\Bigg\}; \tag{6.5}$$

$$\textbf{GLATSB (bootstrap)}:$$
$$\Bigg\{\Big\langle s,\ c,\ b\frac{\sum_{i=1}^{l-1} r_i/r_1}{R}S + \frac{\sum_{i=1}^{l-1}\bar{b}(i)}{R}S + \frac{br_l/r_1}{R}k+$$
$$\frac{\bar{b}(l)(k-1)}{R} + (s-1)\frac{\bar{b}(l)}{S},\ \frac{\bar{b}(l)}{S}\Big\rangle \ \Big|\ \forall s = 1, 2, \ldots, S,$$
$$l = 1, 2, \ldots, C, \text{and } k = 1, 2, \ldots, S\Bigg\}, \tag{6.6}$$

where the four elements are TV channel, device class, burst start time, and burst size, respectively. We notice that the second and the fourth terms of the burst start time in Eq. (6.5) consider the amount of air medium time occupied by the bootstrap bursts. The main difference between Eqs. (6.5) and (6.6) is the last term in Eq. (6.6), which effectively assigns the time between two normal bursts to bootstrap bursts of all S TV channels.

Figure 6.8 shows a simple example of the GLATSB scheme with two TV channels, where each TV channel is scalably coded into two layers. Notice that, the GLATSB scheme adds bootstrap bursts after each normal burst of layer c, where these bootstrap bursts occupy

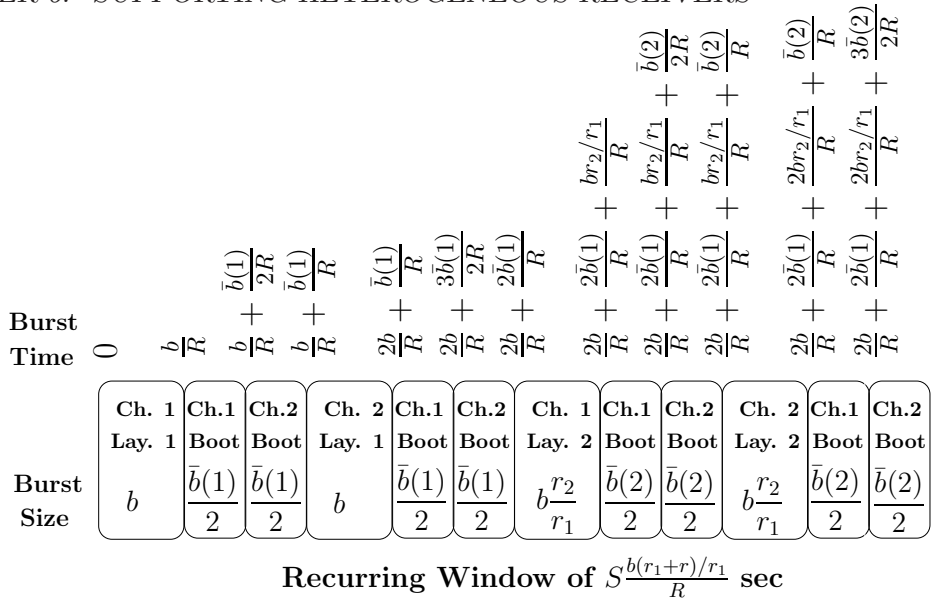


Figure 6.8: An illustrative example of the proposed GLATSB scheme.

the air medium for time $\bar{b}(c)$. This air medium time is equally divided among TV channel, e.g., each bootstrap burst in this example has burst size $\frac{\bar{b}(1)}{2}$ (or $\frac{\bar{b}(2)}{2}$) kb. Last, inserting bootstrap bursts would shift normal bursts to their right, which is taken into considerations in the start time (the third element of the 4-tuple) in Eqs. (6.5) and (6.6).

We mention that the GLATSB scheme is fairly general, as it supports layers with *diverse* bit rates. In contrast, the Layer-Aware Time Slicing with Delay Bound (LATSB) scheme proposed in our earlier work [37] assumes all layers have the same bit rate, i.e., $r_c = r/C$ for all $c = 1, 2, \ldots, C$. Hence, the LATSB scheme is a special case of the newly proposed GLATSB scheme.

6.6.2 Analytical Analysis

We prove the correctness of the proposed GLATSB scheme in the next theorem. We also quantify its performance in the same proof.

Theorem 13. *The GLATSB scheme (Eqs. (6.5) and (6.6)) specifies a feasible time slicing scheme for a recurring window of* $\frac{b(r_1+r)S}{r_1R}$ *sec, where (i) no two bursts overlap with each other, and (ii) bursts are long enough to send data for all mobile devices to playout till the next burst. Furthermore, the energy saving achieved by mobile devices in class* c ($c =$

$1, 2, \ldots, C$*) is given by:*

$$\gamma_c = 1 - \frac{\sum_{i=1}^{c} r_i}{(r_1 + r)S} - \frac{RT_o c r_1}{b(r + r_1)S}. \tag{6.7}$$

The energy saving achieved by mobile devices that receive the bootstrap bursts is:

$$\gamma_b = 1 - \frac{r_1}{(r_1 + r)S} - \frac{RCT_o r}{(r_l + r)b}. \tag{6.8}$$

Finally, the maximal channel switching delay is:

$$d_m = \frac{b(r + r_1) \max_{c=1}^{C} r_c}{Rrr_1}. \tag{6.9}$$

Proof. First, the construction of the GLASTB scheme guarantees that no bursts intersect with each other in time as the air medium of R kbps is fast enough to transmit all bursts within allocated time slots. Second, consider all S bootstrap bursts for an arbitrary TV channel between its normal bursts of layer c and $c + 1$. Each of these bootstrap bursts is $\bar{b}(c)/S$ long, and thus the aggregate burst size is $\bar{b}(c) = \frac{br_c/r}{S}$ kb. Furthermore, the bootstrap bursts after the normal burst of layer $c+1$ arrives $\frac{\bar{b}(c)+br_c/r_1}{R}$ sec later. Because the bootstrap bursts carry the base layer with a bit rate of r_1 kbps, each bootstrap burst can support a playout time of:

$$\frac{br_c/r}{Sr_1} \geq \frac{br_c/(rr_1)}{R/(r + r_1)} = \frac{br_c(\frac{1}{r} + \frac{1}{r_1})}{R} = \frac{\bar{b}(c) + br_c/r_1}{R},$$

where the inequality follows the definition of S (i.e., $S \leq \frac{R}{r+r_1}$). Since the playout time is no shorter than the time period to the next burst, the bootstrap bursts are long enough for smooth playout. We next consider two adjacent normal bursts for the same layer c of a specific TV channel. The burst transmits up to $b\frac{r_c}{r_1}$ kb data, and the time difference between them is $S\frac{b(r_1+r)}{r_1 R}$ sec. Since these two normal bursts carry the layer c with a bit rate of r_c kbps, each such normal burst can support a playout time of:

$$\frac{b(r_1 + r)S}{r_1 R} \leq \frac{b(r_1 + r)R}{r_1 R(r + r_1)} = \frac{b}{r_1} = b\frac{r_c}{r_1},$$

which indicates that the normal bursts are long enough for smooth playout. This proves the correctness of the GLATSB scheme.

Next, following the definition of energy saving, the energy saved for mobile devices that receive c layer is:

$$\gamma_c = 1 - \frac{\frac{b\sum_{i=1}^{c} r_i/r_1}{R} + cT_o}{S\frac{b(r_1+r)}{r_1 R}},$$

and the energy saved for mobile devices that receive bootstrap bursts is:

$$\gamma_b = 1 - \frac{b/R + CST_o}{S\frac{b(r_1+r)}{r_1 R}}.$$

In these two equations, the first term of the numerator is the time to receive actual video data, the second term of it accounts for the total overhead duration, and the denominator is the recurring window size. Simplifying these two equations leads to Eqs. (6.7) and (6.8).

Finally, the channel switching delay is the maximal time difference between two bootstrap bursts for the same TV channel. Note that, the time difference between two bootstrap bursts after a layer c normal burst is $\frac{\bar{b}(c)+br_c/r_1}{R}$, where the first term in the numerator accounts for the bootstrap bursts and the second term of it accounts for the normal burst. Hence, the maximal channel switching delay is:

$$d_m = \frac{\max_{c=1}^{C}[\bar{b}(c) + br_c/r_1]}{R} = b\frac{(r+r_1)\max_{c=1}^{C} r_c}{Rrr_1},$$

which leads to Eq. (6.9). □

The next corollary enables network operators to bound the channel switching delay precisely. This is a direct result of Eq. (6.9).

Corollary 3. *For a given maximal channel switching delay d_m, applying the GLATSB scheme (Eqs. (6.5) and (6.6)) with any normal burst size b, $b \leq b_m$ guarantees that mobile devices never suffer from switching delay longer than d_m, where $b_m = d_m \frac{Rrr_1}{(r+r_1)\max_{c=1}^{C} r_c}$.*

The following corollary is a direct result of Theorem 13.

Corollary 4. *Consider a special case of the GLATSB scheme, where the layers have uniform bit rates, i.e., $r_c = r/C$ for all $c = 1, 2, \ldots, C$. The energy saving achieved by mobile devices in class c ($c = 1, 2, \ldots, C$) is given by:*

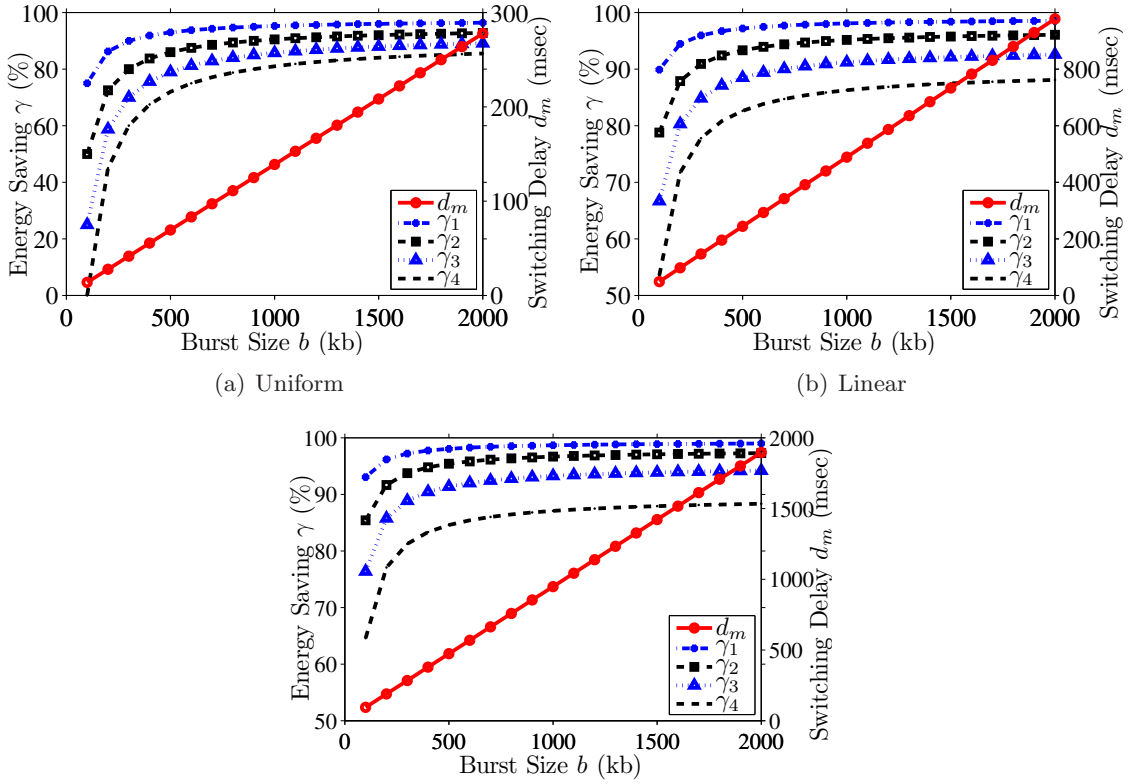
$$\gamma_c = 1 - \frac{c}{(C+1)S} - \frac{RT_o c}{b(C+1)S}.$$

The energy saving achieved by mobile devices that receive the bootstrap bursts is:

$$\gamma_b = 1 - \frac{1}{(C+1)S} - \frac{RCT_o}{(C+1)b}.$$

Finally, the maximal channel switching delay is:

$$d_m = \frac{b + b/C}{R}.$$



(a) Uniform

(b) Linear

(c) Exponential

Figure 6.9: Diverse energy saving supported by the GLATSB scheme, and resulting channel switching delay.

This corollary considers simplified scalable streams where layers have uniform bit rates, and quantifies the performance of the GLATSB scheme under such assumption. We notice that we call this simplified scheme as Layer-Aware Time Slicing with Delay Bound (LATSB) scheme in our previous results [37], and this corollary is consistent with our earlier analytical analysis presented in [37].

6.6.3 Numerical Analysis and Discussion

We apply the same network parameters used in Section 6.5.3 to numerically study the performance of the GLATSB scheme. Since the GLATSB scheme supports layers with diverse bit rates, we consider three different scenarios to assign bit rate to individual layers, including uniform, linear, and exponential. Specific definition on these scenarios are given

in Section 6.5.3. We vary the base layer burst size b, and we compute the energy saving and channel switching delay of mobile devices in different classes using formulas derived in Theorem 13.

We plot the results in Figure 6.9, which shows that the GLATSB scheme enables mobile devices to achieve a wide range of energy saving values. More importantly, this figure reveals that, compared to the GLATS scheme, the GLATSB scheme dramatically reduces the channel switching delay: less than 300 msec delay can be achieved in the uniform scenario, and the delays never exceed 2 sec in all considered scenarios. This means the GLATSB scheme does achieve its goal to reduce channel switching delays. We mention that, in this analysis, we assume mobile devices receive bootstrap bursts for short, *transient*, time periods, and only consider mobile devices that receive normal bursts. We eliminate this assumption when conducting the the empirical evaluation in Section 6.7.

Last, we notice that low channel switching delays achieved by GLATSB scheme comes at an expense of bandwidth overhead of Sr_1 kbps, since the base layers are transmitted *twice*. This overhead is, however, controllable: network operators may decide to use a lower base layer rate. Smaller base layer not only mitigates bandwidth overhead, but also supports more heterogeneous mobile devices. This is because mobile devices that are not capable to receive and render the base layer are effectively excluded from the mobile TV service.

6.7 Evaluation on a Mobile TV Testbed

We evaluate the proposed broadcasting schemes using a real mobile TV testbed. We first describe the testbed and experimental setup. We then present the results.

6.7.1 Setup

We use a real mobile TV testbed presented in Section 3.6 to evaluate our proposed schemes. We have also implemented both the GLATS and GLATSB schemes in the testbed. We notice that, to the best of our knowledge, there exists no other broadcast schemes for broadcasting scalable video streams in current systems. For comparison, we implement the single service (SS) and parallel service (PS) broadcast schemes presented in Secs. 6.4.1 and 6.4.2, respectively. These two schemes work in current systems, and achieve the same energy saving and channel switching delay. Hence, we denote them as CUR in the figures. Using the H.264/SVC [51] reference software [97], we have encoded the City video sequence, into four

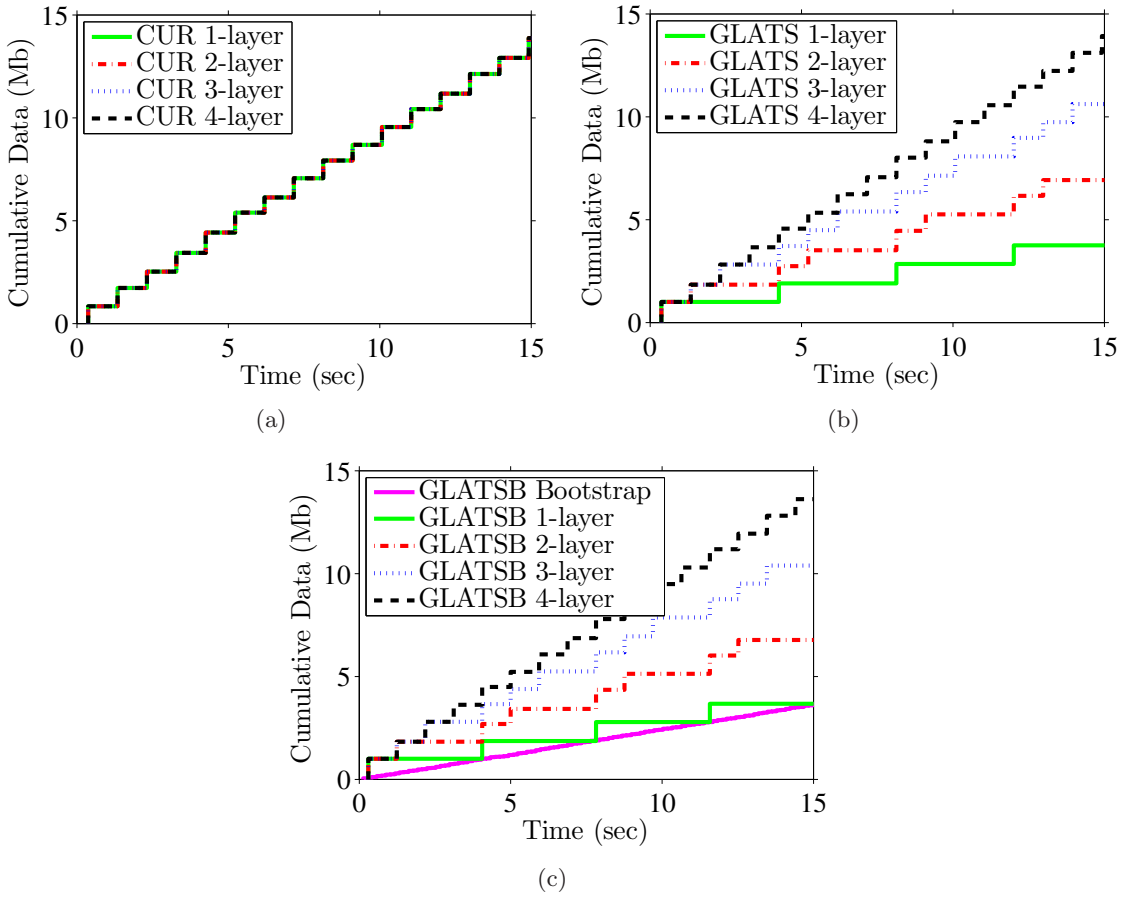


Figure 6.10: Cumulative received data: (a) CUR, (b) GLATS, and (C) GLATSB.

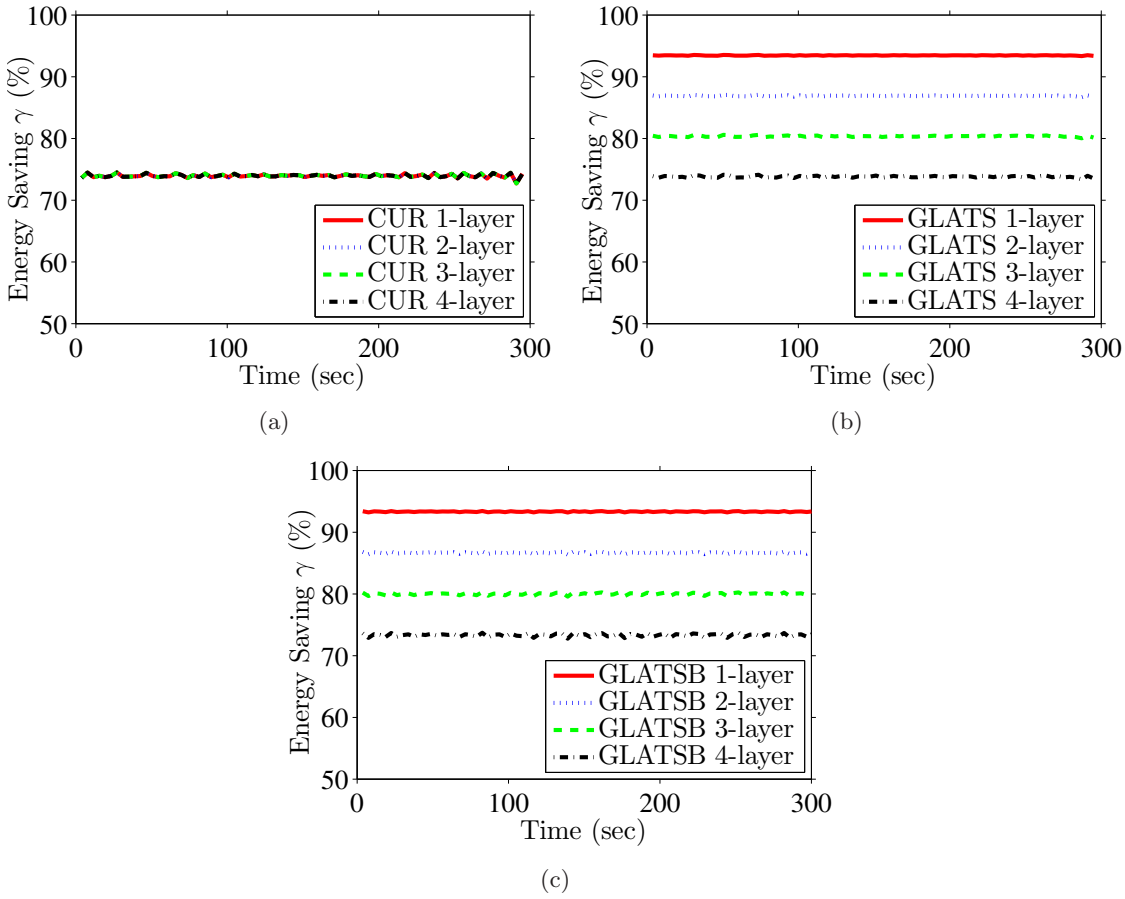


Figure 6.11: Sample energy saving achieved by a mobile device in each classes: (a) CUR, (b) GLATS, and (C) GLATSB.

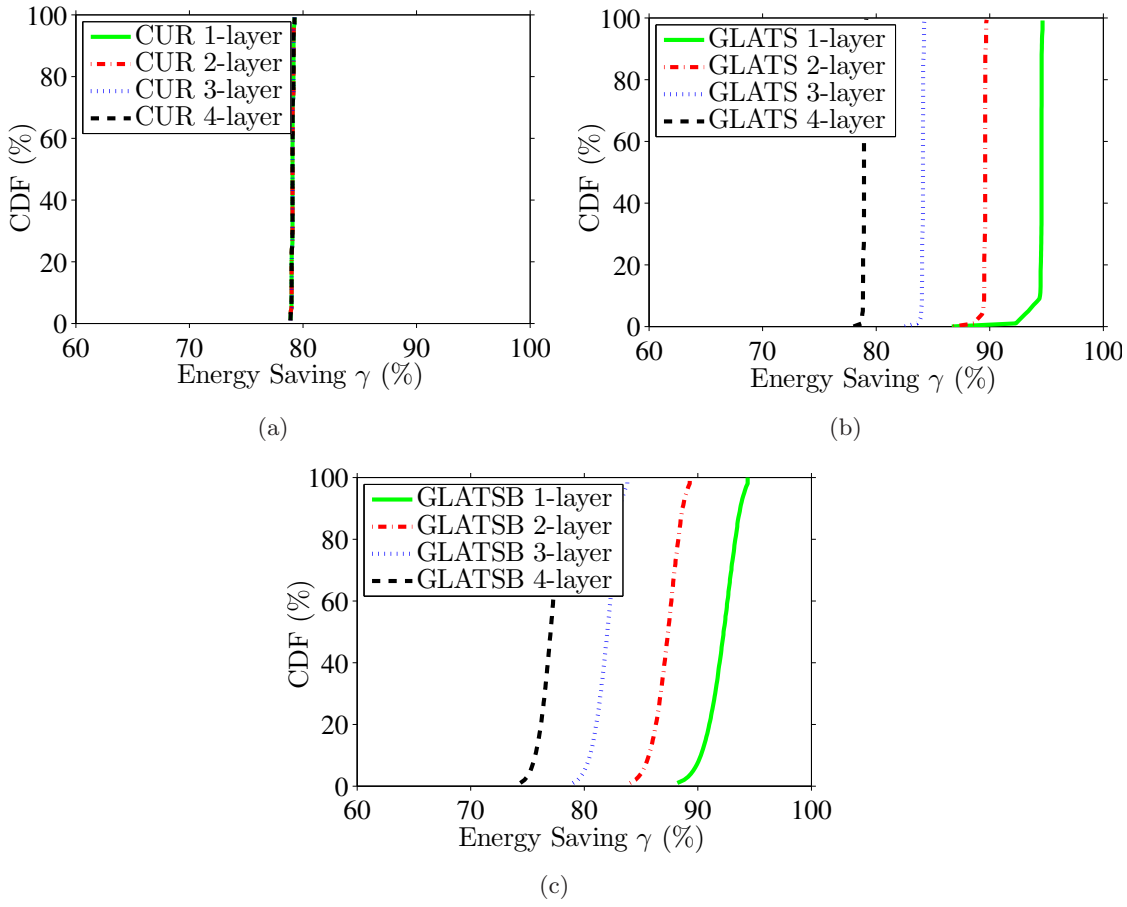


Figure 6.12: Energy saving achieved by (all) mobile devices in different classes: (a) CUR, (b) GLATS, and (C) GLATSB.

SNR scalability layers, where each layer has a bit rate of 192 kbps and is in CIF (352x288) resolution at 30 fps frame rate, Given that the City sequence is quite short, we concatenated it several times to form a 5-min video sequence. We broadcast the same sequence in all TV channels, because different video characteristics do not significantly affect the quality of service metrics considered in this section. Furthermore, as there is no effective rate control algorithms built in the H.264/SVC reference software [97], we had to search for appropriate quantization parameter (QP) values by encoding the same video many times. This is very time consuming even for a single video sequence, e.g., encoding a 10-sec sequence could take more than 12 hours in our experience. With the coded stream, we configured the modulator card to use 8 MHz bandwidth, QPSK (quadrature phase-shift keying) modulation, 3/4 code ratio, and 1/8 guard interval. This leads to the channel bandwidth of 8.289 Mbps [46]. We broadcast 4 TV channels for 5 min using the GLATS and GLATSB schemes, and repeated the same test using the CUR scheme.

We have instrumented the testbed to save log files for offline analysis. The log files contain start and end times of each burst, its size, and the layer it belongs to. Using these logs, we wrote a software utility to emulate the channel switching behavior of a large number (1 million) of mobile devices. We generate random channel switching events using Bernoulli trials. For every mobile device, we toss a biased coin every second and issue a channel switching command if the trial is success. The new selected channel is randomly chosen from all broadcast channels other than the currently watched one. We configured the probability of success to vary the average watch time of each channel from 1 sec to 60 sec. We also varied the burst size from 500 to 1500 kb. If not otherwise specified, we present sample results for 60-sec average watch time, and 1000 kb burst size.

We ran the simulation against every log file collected from the testbed, and we computed the channel switching delay d and energy saving γ. We measured the channel switching delay by searching for the next burst of the selected TV channel and computing the time difference between it and the channel switching event. We let the overhead duration $T_o = 100$ msec, and measured the energy saving by calculating the fraction of time that the receiving circuit is on between every two channel switching events. We emphasize that when computing the energy saving for the GLATSB scheme, we divide the watching period into two parts: when a device receives bootstrap bursts, and when it receives normal bursts. We calculated the energy saving in both periods and reported the weighted average of them. Since we consider both time periods in the experiments, we no longer assume that mobile devices

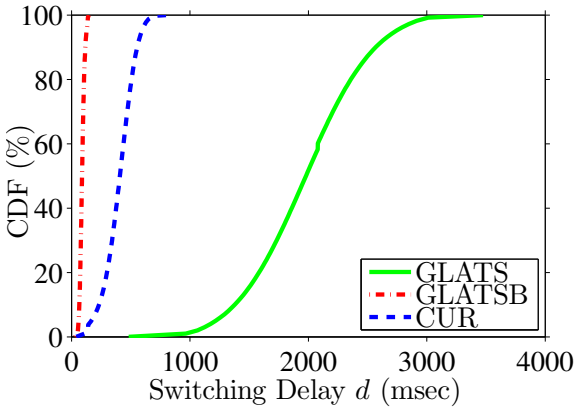


Figure 6.13: Delay of different schemes.

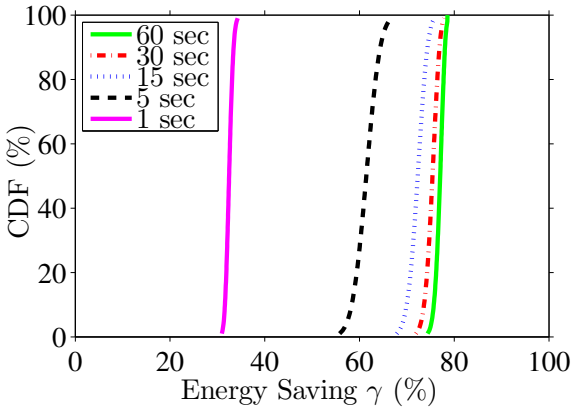


Figure 6.14: Implication of watch time.

receive bootstrap bursts only for a transient time period as we did in Section 6.6.3.

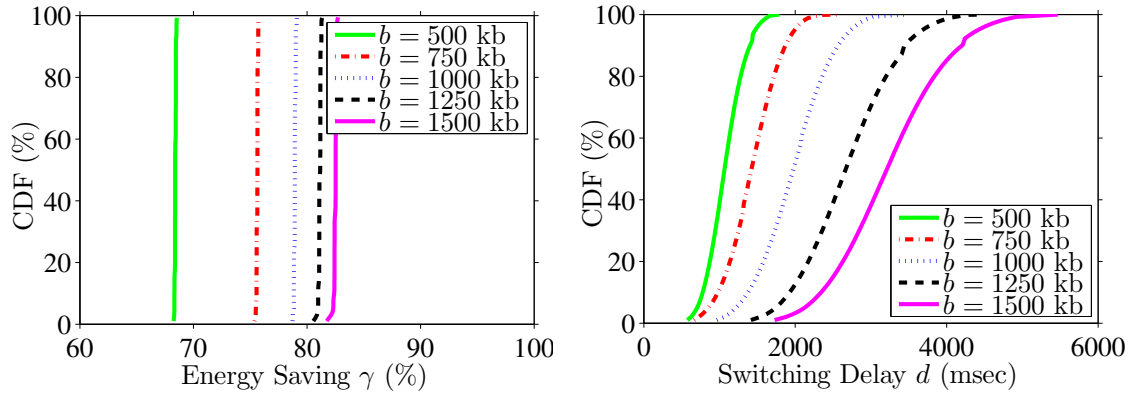
6.7.2 Results

We present sample results for TV channel 1, while results for other TV channels are similar.

Cumulative Data Dynamics: We first plot the cumulative received data in Figure 6.10 for a 15-sec time period. In this figure, every staircase step represents a received burst. This figure reveals that the CUR scheme does not allow mobile devices to receive substreams that are smaller than the complete video streams, while both the GLATS and GLATSB schemes enable mobile devices to skip those bursts that are of no use to them. In addition Figure 6.10(c) illustrates that in GLATSB, bootstrap bursts come more often, and the accumulated bit rate of bootstrap bursts is equivalent to that of the base layer.[2] This shows that mobile devices who receive bootstrap bursts can quickly render the video at the base layer quality.

Diverse energy saving: We first plot the energy saving achieved by a sample mobile device in Figure 6.11. We then compute the average energy saving of all mobile devices, and report the CDF curves in Figure 6.12. These two figures show that the CUR scheme leads to no energy saving differentiation, while the GLATS and GLATSB schemes enable proportional energy saving for mobile devices in different classes. This confirms our earlier

[2]The line of bootstrap bursts consists of staircases, but in smaller scale.



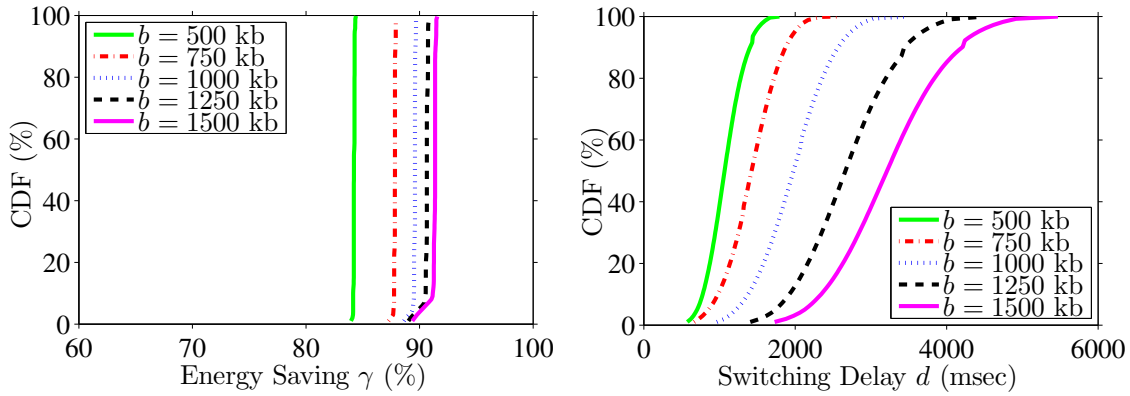
(a) GLATS, energy saving

(b) GLATS, channel switching delay

(c) GLATSB, energy saving

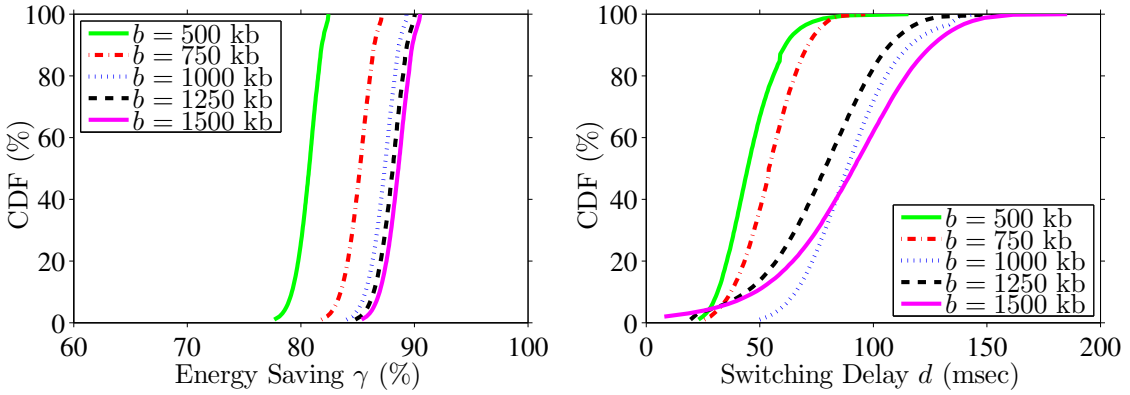
(d) GLATSB, channel switching delay

Figure 6.15: Implication of burst size on energy saving and channel switching delay: mobile devices that receive all 4 layers.



(a) GLATS, energy saving

(b) GLATS, channel switching delay



(c) GLATSB, energy saving

(d) GLATSB, channel switching delay

Figure 6.16: Implication of burst size on energy saving and channel switching delay: mobile devices that receive 2 layers.

observations (in Section 6.4.4) that broadcasting scalable streams using CUR scheme cannot support heterogeneous mobile devices.

Channel Switching Delay: We plot the CDF curves of channel switching delays for all considered schemes in Figure 6.13. This figure shows that although GLATS scheme supports mobile devices with heterogeneous resources, it may lead to high channel switching delay: up to 4-sec delay is observed in this experiment. In contrast, GLATSB scheme results in *negligible* channel switching delay: about 200 msec is achieved.

Implication of burst size: We next vary burst size b from 500 to 1500 kb. This covers the whole practical range of b values, since 1565 kb is the maximal burst size specified by DVB-H standard documents [16]. We first present results for the mobile devices that receive the complete streams. We plot the CDF curves of energy saving and channel switching delay in Figure 6.15. Figure 6.15(a) shows that increasing burst size allows the GLATS scheme to achieve higher energy saving. However, Figure 6.15(b) reveals that larger burst size also increases the channel switching delay: letting $b = 1500$ kb leads to as high as 6-sec delay. These two figures show that the GLATS scheme uses b to control the tradeoff between energy saving and channel switching delay, which is inefficient. In contrast, Figs. 6.15(c) and 6.15(d) show that increasing b in the GLATSB scheme also leads to higher energy saving, however, it does not result in excessive channel switching delay: the delay is shorter than 200 msec in all cases. Figure 6.16 shows the results for mobile devices that receive 2 layers. The observations made in Figure 6.15 are also applicable to this figure. This confirms that the above discussions are true for all mobile devices despite how many layers they receive and decode.

Implication of per channel watch time: In the GLATSB scheme, mobile devices that receive bootstrap bursts incur lower energy saving. Hence, the frequency of channel switching events can affect the average energy saving. To quantify this impact, we vary the time that a user would watch a TV channel from 1 sec to 60 sec. We plot the CDF curves of average energy saving of mobile devices that receive the complete streams in Figure 6.14. This figure shows that frequent channel switching events only degrade the energy saving in extreme cases, in which users constantly change TV channels.

6.8 Conclusions

In this chapter, we studied the problem of broadcasting scalable video streams over a shared air medium to mobile devices with *heterogeneous* resources, such that mobile devices can render the most appropriate substreams while achieving high energy saving and low channel switching delay. We analyzed current mobile video broadcast networks, and analytically showed that they cannot efficiently broadcast scalable coded streams. We then proposed two scalable broadcast schemes: GLATS and GLATSB. We formally proved the correctness of the proposed schemes, and we analytically quantified their performance in terms of energy saving and channel switching delay. The proposed schemes can be implemented in current base stations, and they produce bursts of traffic that are compliant with current mobile video broadcasting standards such as DVB-H. The main difference between GLATS and GLATSB is that the latter ensures very small channel switching delays, but at a small cost of lower bandwidth utilization. We implemented the GLATS and GLATSB schemes in a real mobile TV testbed to show their practicality and efficiency. Our extensive experiments showed that both the GLATS and GLATSB schemes enable energy saving differentiation: between 75% and 95% were observed. Moreover, the GLATSB scheme also achieves low channel switching delays: 200 msec is possible with typical system parameters.

Chapter 7

Conclusions and Future Work

In this chapter, we first summarize our contributions of this thesis. We then give network operators some guidelines on choosing the solutions suitable for their mobile video broadcast networks. Finally, we outline several future research directions.

7.1 Conclusions

We studied mobile video broadcast networks in which a base station concurrently transmits multiple video streams over a shared air medium to many mobile devices. Such mobile networks support one-to-many multicast/broadcast communications and achieve high bandwidth efficiency because a copy of video stream can be received by all mobile devices within the range of the base station. These mobile networks are expected to attract millions of subscribers worldwide. The goal of this thesis is to improve the efficiency and quality-of-services of these networks. The thesis presented systematic and provably optimal (or very close to optimal) algorithms to: (i) maximize streaming quality, (ii) maximize energy saving, (iii) maximize bandwidth efficiency, (vi) efficiently control channel switching delay, and (v) support heterogeneous mobile deices. All proposed algorithms are practical and can run in real time, as verified by our implementation in actual mobile video streaming testbed that conforms to one of the most common international standard (DVB-H). Therefore, we believe that the research contributions of this thesis will have significant practical impacts on mobile video streaming networks and will provide millions of mobile users much better quality-of-service in terms of better perceived video quality, longer viewing time, shorter channel switching delay, and more flexibility in customizing video streams to better match

users' needs and the capacity of their mobile devices.

Since mobile devices are battery powered, energy conservation is critical for them to achieve long viewing time before users have to replace or recharge batteries. Therefore, modern broadcast network standards, such as DVB-H and MediaFLO, dictate energy saving mechanism in order to prolong viewing times. The most common technique for saving energy is to broadcast each video stream in bursts at a bit rate much higher than the encoding bit rate of that stream. Mobile devices, therefore, can receive a burst of video data and put their receiving circuits into sleep to save energy. This is called time slicing. In Chapter 3, we considered the problem of maximizing energy saving of mobile devices in a mobile video broadcast network, in which a base station concurrently transmits multiple Constant-Bit-Rate (CBR) streams to many users. To prevent playout glitches and achieve the optimum streaming quality, the base station must carefully construct burst schedules that result in: (i) no buffer violations, (ii) no burst conflicts, and (iii) high energy saving. We considered a very general problem in which every video stream can be coded at a difference bit rate. We formulated this problem as an optimization problem, and we formally proved that it is NP-complete. We then solved this problem in two steps. First, we proposed a practical simplification, which allows video streams to be classified into multiple classes with power-of-two bit rate increments. We proposed a burst scheduling algorithm, called P2OPT (Power-of-Two Optimal), to solve this practical simplification. We analytically proved that the P2OPT algorithm is optimal in terms of energy saving and runs in polynomial time. We used simulations and experiments to validate the correctness and optimality of the P2OPT algorithm. The experimental results from a real mobile TV testbed indicate that the P2OPT algorithm achieves optimal energy saving: savings between 77% to 99% were observed for streams at different bit rates. In addition, we empirically showed that, with only six possible bit rates, using the P2OPT algorithm to broadcast 10 videos can achieve vary small quality variation among videos: less than 1 dB was observed in our experiments. The P2OPT algorithm is suitable for network operators that encode videos in CBR streams and would trade strict video quality fairness for optimal energy saving.

In Chapter 3, we also solved the general form of the energy optimization problem in which CBR streams can have arbitrary bit rates. Since the energy optimization problem is NP-complete, we proposed an approximation algorithm, called DBS (Double Buffering Scheduling), to solve this general form formulation. We showed the correctness and efficiency of the DBS algorithm. More importantly, we analytically gave an approximation factor of

the DBS algorithm, and we numerically showed that the approximation factor is less than 5% under typical network parameters. This means that the DBS algorithm is near-optimal in terms of energy saving. We implemented the DBS algorithm in a simulator as well as in a real mobile TV testbed. Our simulation and experimental results indicate that the DBS algorithm runs efficiently and achieves high energy saving. For example, the DBS algorithm achieves an energy saving no worse than 7% lower than a conservative upper bound of energy saving. Such high energy savings were achieved without incurring high time complexity: the DBS algorithm runs in real-time. The DBS algorithm is suitable for network operators that encode videos in CBR streams and require fairness in terms of quality among video streams.

In Chapter 4, we considered the problem of constructing burst schedules for a mobile video broadcast network in which a base station concurrently transmits multiple Variable-Bit-Rate (VBR) streams to many users, such that the energy saving of mobile devices and the goodput of the network are both maximized. Unlike CBR streams that impose a constant workload on the broadcast network, transmitting VBR streams may lead to dynamic aggregate bit rates that might exceed the network bandwidth. We considered two type of mobile video broadcast networks: closed-loop networks, in which all video streams are jointly encoded to ensure the aggregate bit rate does not exceed the network bandwidth, and open-loop networks, in which video streams are independently encoded and may occasionally overload the broadcast network. We formulated the problem of maximizing energy saving and maximizing goodput as an multiobjective optimization problem, and we showed that it is NP-complete. We then proposed an approximation algorithm, called SMS (Statistical Multiplexing Scheduling), to solve this problem. The SMS algorithm only requires a small lookahead window (in the order of a few seconds) for frame sizes, and thus is an online algorithm. We showed that the correctness of the SMS algorithm, and proved that it is optimal in terms of goodput and near-optimal in terms of energy saving. Moreover, the SMS algorithm runs in polynomial time. Most importantly, we analytically gave the approximation gap of the SMS algorithm and numerically showed the approximation gap is less than 1.5% under typical network parameters.

We showed that the SMS algorithm produces glitch-free burst schedules in closed-loop networks, and thus it is suitable for network operators of closed-loop networks. We also argued that the SMS algorithm minimizes the number of glitches in open-loop networks, and can also be used by network operators of open-loop networks. However, the SMS algorithm might lead to playout glitches in open-loop networks. We studied the open-loop networks

that have longer lookahead windows, and proposed another burst scheduling algorithm, called SMS′, to reduce the number of playout glitches due to burstiness of aggregate bit rates. The SMS′ algorithm utilizes the slack time of the air medium and tries to send some video data in advance in order to absorb the rate burstiness. Hence, the SMS′ algorithm is suitable for open-loop networks of recorded videos where longer lookahead windows are possible, while the SMS algorithm is suitable for open-loop networks of live videos where only a short lookahead window is feasible. We implemented both SMS and SMS′ algorithms in a trace-driven simulator, and compared their performance against the burst scheduling algorithms currently used in commercial base stations. The simulation results indicate that the SMS and SMS′ algorithms outperform current burst scheduling algorithms from several aspects. More specifically, the SMS algorithm produces burst schedules that lead to: (i) fewer missed frames, (ii) more concurrent video streams, and (iii) higher energy saving. Moreover, compared to the SMS algorithm, the SMS′ algorithm effectively reduces the number of playout glitches without sacrificing energy saving. We also implemented the SMS algorithm in a real mobile TV testbed to show its practicality and efficiency. The results from this testbed confirmed that the SMS algorithm: (i) runs efficiently, (ii) achieves high energy saving, and (iii) result in no playout glitches.

In Chapter 5, we considered the problem of constructing time slicing schemes to control channel switching delay, such that the channel switching delay from any channel to any other channel never exceeds a maximum switching delay specified by network operators. In addition, the broadcast schemes should also achieve high energy saving, which is critical for mobile devices. We first analyzed the time slicing scheme currently used in deployed mobile video broadcast networks, and we showed that it achieves low energy saving when short channel switching delays are required. That is, the current time slicing scheme cannot efficiently control the channel switching delay. We then proposed three time slicing schemes for network operators who need to control the channel switching delays: SIMU, SIMU-S and SIMU-S+. The SIMU scheme is suitable in legacy environment where scalable video decoders are not available on mobile devices. The SIMU-S scheme is suitable for mobile video broadcast networks with some idle bandwidth, while the SIMU-S+ scheme is for bandwidth saturated networks. We proved the correctness of these time slicing schemes, and we analytically analyze their performance. More importantly, we implemented the proposed time slicing schemes in a real mobile TV testbed. The experimental results from the testbed showed that the proposed time slicing schemes prevent mobile devices from exceeding the

target channel switching delay, while achieving much higher energy saving compared to the time slicing scheme currently used in base stations. For example, in our experiments, the SIMU-S scheme achieves up to 93% energy saving, while the current scheme only results in 74%.

In Chapter 6, we studied the problem of encapsulating multiple scalable coded streams into bursts and broadcasting these bursts to many mobile devices in an appropriate way so that a mobile device may opt for receiving a lower quality substream in order to save energy and prolong watch time. We first showed that existing base stations cannot efficiently broadcast scalable video streams, because, without properly organizing video data in bursts, mobile devices that receive a lower quality substream do not achieve higher energy saving. We then proposed two time slicing schemes for network operators who need to support heterogeneous mobile devices. The GLATS (Generalized Layer-Aware Time Slicing) scheme allocates a burst for each layer of every video stream, so that mobile devices can selectively receive relevant bursts to achieve proportional energy saving. The GLATSB (Generalized Layer-Aware Time Slicing with Delay Bound) scheme employs the idea used in SIMU/SIMU-S schemes and can efficiently control the channel switching delay. The GLATS algorithm is suitable for network operators that want to support heterogeneous mobile devices, but do not have stringent switching delay requirements. For network operators that want to trade some network bandwidth for controlling channel switching delay, the GLATSB algorithm is more appropriate. We proved the correctness of the GLATS and GLATSB algorithms, and we analytically quantify their performance. More importantly, we implemented the GLATS and GLATSB algorithms in a real mobile TV testbed. The experimental results from the testbed reveal that both algorithms allow users to opt for the most appropriate video quality in order to save more energy, while the GLATSB algorithm also prevents mobile devices from exceeding the target channel switching delay. The diverse energy saving and short channel switching delays are not possible in the time slicing scheme currently used in base stations.

Last, we summarize the proposed burst scheduling algorithms and time slicing schemes in Table 7.1. In the same table, we also give recommendations to network operators for choosing the solution most suitable to their mobile video broadcast networks. Figure 7.1 further breaks down the recommendations into three main *features*: device heterogeneity, bounding switching delay, and high bandwidth efficiency. Network operators may follow this figure and answer up to three questions in order to find the most suitable solutions for

Table 7.1: List of the proposed solutions and recommendations for network operators.

Solution	**Suitable Broadcast Networks**
P2OPT	CBR streams, trade channel quality variation for optimal energy saving
DBS	CBR streams, no channel quality variation, near-optimal energy saving
SMS	VBR streams, closed-loop networks
	VBR streams, open-loop networks, live videos
SMS′	VBR streams, open-loop networks, recorded videos
SIMU	controlling switching delay, legacy mobile devices
SIMU-S	controlling switching delay, mobile devices with scalable decoders
SIMU-S+	controlling switching delay, saturated network bandwidth
GLATS	support heterogeneity, no switching delay requirements
GLATSB	support heterogeneity, trade bandwidth for controlling switching delays

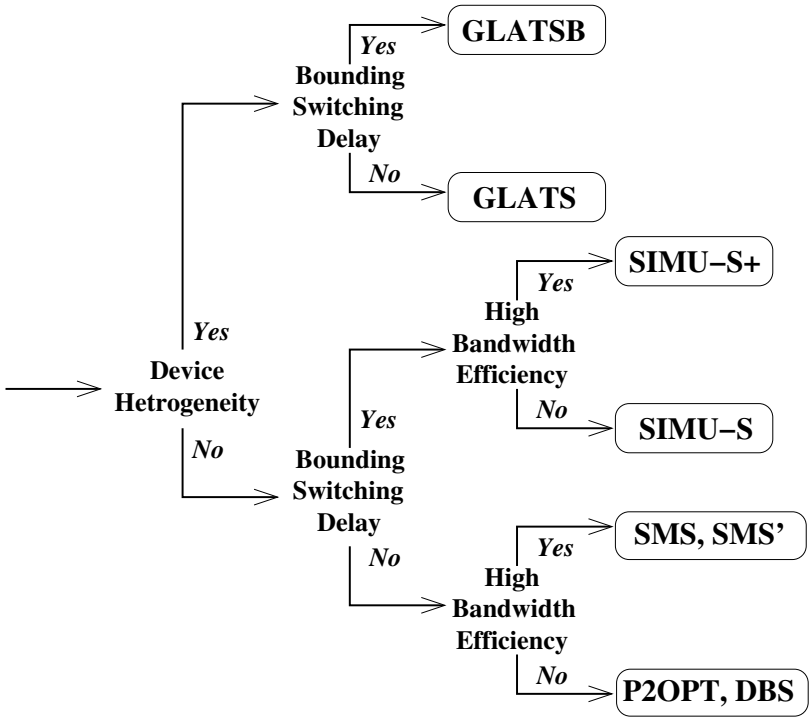


Figure 7.1: Questionnaire to guide network operators to choose the most suitable burst scheduling algorithms.

them.

7.2 Future Work

The problems studied in this thesis can be extended in several directions. This section presents a number of possible extensions for future research.

7.2.1 Statistical Multiplexing of Live and Recorded Video Streams

In Chapter 4, we study the problem of statistical multiplexing of multiple VBR streams in order to minimize the number of playout glitches and maximize the energy saving. We propose the SMS′ algorithm for open-loop networks to reduce the number of playout glitches caused by overloading the broadcast network, which happens when the total bit rate of all video streams exceeds the network bandwidth. Compared to the SMS algorithm, the SMS′ algorithm employs a longer lookahead window in order to identify the spikes of the total bit rate, and then utilizes the slack time of the air medium to transmit some video data in advance if the bit rate spikes will overload the broadcast network. The SMS′ algorithm is more suitable for recorded video streams because these streams have long lookahead windows for the frame size. In contrast, broadcast networks with live video streams may need to resort to the SMS algorithm if only a small lookahead window is available, and thus are vulnerable to playout glitches.

Typical video broadcast networks, however, consist of many recorded programs, such as movies and TV episodes, and a few live programs, such as TV news and sports games. Since a longer lookahead window may not be available in live programs, typical broadcast networks may not benefit from the SMS′ algorithm. Hence, enhancing the proposed SMS′ algorithm to support mobile video broadcast networks with a mixture of recorded and live programs will improve its applicability. There are several ways to enhance the SMS′ algorithm. For example, we may use longer lookahead windows on recorded programs to know their bit rate requirements and employ VBR video traffic models, such as [118], on live programs to predict their bit rate requirements, in order to identify spikes of the total bit rate. We may then utilize any slack time of the air medium to absorb the bit rate spikes, so that the number of playout glitches is minimized.

7.2.2 Time Slicing in other Mobile Video Broadcast Networks

Although we use DVB-H networks as the sample mobile video broadcast network throughout this thesis, the proposed solutions can be readily applied to many other broadcast networks. In particular, similar to DVB-H, several broadcast standards also encapsulate broadcast stream in MPEG-2 TS format, which include T-DMB, ISDB-T, A-VSB (Advanced Vestigial SideBand, developed in USA), DMB-T/H (Digital Terrestrial Multimedia Broadcast, developed in China), and CDMB (China's Digital Multimedia Broadcasting) [119]. These networks sequentially transmit the fixed-size packets of MPEG-2 TS streams over the air medium, which enables the base station to construct logical channels using Time Division Multiplex (TDM) and implement time slicing in order to save energy on mobile devices. Therefore, our proposed solutions can be directly used in these mobile video broadcast networks to improve streaming quality and bandwidth efficiency.

Although MediaFLO (described in Section 2.1.3) employs a new multiplexing format that is different from MPEG-2 TS, this multiplexing format also consists of fixed-size packets organized into logical channels called Multicast Logical Channels (MLCs) [119]. MediaFLO divides the whole wireless spectrum into 4000 subcarriers and it aggregates these subcarriers into eight *interlaces*, where each interlace spans over the whole wireless spectrum for higher frequency diversity. MediaFLO supports time slicing that is more sophisticated than other broadcast networks: each MLC is assigned a time slot in the granularity of an OFDM symbol, and a frequency slot in the granularity of an interlace. Similar to DVB-H, MediaFLO base stations embed the time and frequency slots of the next burst in the headers of the current burst, and mobile devices turn off their receiving circuits between bursts to save energy.

While our solutions are applicable to MediaFLO networks, they may not fully utilize the additional flexibility provided by the frequency slots. Enhancing our solutions for MediaFLO networks will allow us to achieve even better performance by capitalizing on the additional flexibility. The new solutions, designed for MediaFLO, will also be useful for other wireless networks developed in the future. This is because OFDM has been widely used in many modern wireless network standards, and these networks may support multiplexing along both the time and frequency axes.

7.2.3 Integrated Multiobjective Scheduling Framework

Although we outline the guidelines, in Section 7.1, for network operators to choose burst scheduling algorithms that are most suitable to their needs, it is desired to have a complete burst scheduling framework that can dynamically *adapt* to network operators' requirements at run-time. This framework should concurrently support high energy saving, high goodput, controlled channel switching delays, and device heterogeneity, but also allows network operators to *drop* some of these capabilities if they are not needed. We call this new scheduling framework as *Integrated Multiobjective Scheduling Framework*, or IMS framework.

The IMS framework borrows ideas and experiences gathered during designing the following three burst scheduling algorithms: SIMU (Chapter 5) for controlling channel switching delays, GLATS (Chapter 6) for supporting heterogeneous mobile devices, and SMS (Chapter 4) for scheduling bursts to achieve optimal goodput of the broadcast network and near-optimal energy saving on mobile devices. More specifically, similar to the SIMU algorithm, the IMS framework reserves $x\%$ of bandwidth for bootstrap trains in order to keep channel switching delays shorter than a $d_{\max}$ value specified by network operators. The residue bandwidth is then used by primary trains, which contain L scalable layers, where $L \geq 1$. The IMS framework is similar to GLATS algorithm in the sense that data from different layers are grouped into different bursts. Last, the IMS framework uses the idea of the SMS algorithm to schedule the primary bursts. This allows network operators to maximize the goodput of the network and energy saving of mobile devices.

We notice that, the IMS framework takes the layer dependency into consideration and sequentially allocates bandwidth to layers from lower- to higher-layers. Therefore, all received data in the IMS framework are *decodable* on mobile devices. Last, the IMS framework is very flexible and can meet various needs from network operators. For example, network operators may choose number of scalable layers L depending on the number of device models supported in their broadcast networks. They can even set $L = 1$ if there exists no heterogeneity. Similarly, network operators may set $x = 0\%$ if controlled channel switching delays are not required. Using the IMS framework allows network operators to achieve almost all merits they could get from any of the burst scheduling algorithms proposed in this thesis. More importantly, network operators can adjust various tradeoffs by configuring the system parameters of the IMS framework, rather than implementing several scheduling algorithms in their base stations.

7.2.4 Transmission Scheduling in Hybrid Cellular-Broadcast Networks for VoD Services

In Section 1.1, we mention that most of our solutions for burst scheduling in broadcast networks can be integrated with techniques such as periodic broadcast to support VoD services. Providing VoD services using broadcast networks, however, may consume too much bandwidth as each video stream is carried by several broadcast channels or multicast groups. To minimize bandwidth requirements, depending on the number of receivers for individual video streams, some videos may better be broadcast/multicast, while others may be more suitable to be unicast. For illustration, consider a video stream requested by one VoD receiver, devoting several broadcast channels to it is *not* bandwidth efficient. In such scenario, streaming using unicast in 3G cellular networks is more efficient because more broadcast channels can be used to send other videos that are received by many more receivers. Therefore, hybrid cellular-broadcast networks for VoD services could lead to better bandwidth efficiency, and thus become more scalable than 3G cellular networks or broadcast networks. Several works in the literature investigate the potential benefits of building such hybrid networks, e.g., Bria studies hybrid networks from the aspect of delivery cost reduction [120]. Transmission scheduling in such hybrid networks, however, is challenging and has not been fully addressed. While a recent work presents a scheduling algorithm for VoD services over multicast-enabled IP networks [121], transmission scheduling over wireless networks to mobile devices is more difficult as we showed in this thesis. Hence, designing an effective and efficient scheduling algorithm for hybrid networks is one of our future works.

7.2.5 Quality-Power Adaptation Framework for Mobile Video Streaming

In Chapter 6, we propose time slicing schemes to efficiently broadcast scalable coded streams, and we show that mobile devices may receive substreams of the full-quality stream in order to save more energy. While the proposed GLATS and GLATSB algorithms allow users to trade video quality for higher energy saving, it may not be easy for users to determine the appropriate version of substream to receive. In fact, the most appropriate version of substream depends on the device capability, the target energy consumption level, and user preferences. To help users on making decisions, we may design a quality-power adaptation framework to systematically control the perceived video quality and the length of viewing time on battery-powered mobile devices. The framework should be general, and it may be

used for standalone receivers (e.g., DVD players and notebooks) as well as mobile receivers obtaining video signals from wireless networks (e.g., mobile TV and video streaming over WiMAX). Furthermore, the framework should support both live streaming (e.g., live TV shows) and pre-encoded streams (e.g., DVD movies).

The framework consists of several quantitative models that map basic energy consumption and video bit rate to intuitive, and easy-to-understand, performance metrics such as the length of viewing time in hours and expected perceived quality in MOS (mean opinion score). To illustrate the potential of this framework, consider a user, Amy, who wants to watch a 30-min TV episode using her cellular phone that only has remaining battery capacity for watching the show for 25 min. Most of current mobile devices cannot *adapt* to the energy constraint, because video streams coded by nonscalable coders must be received and decoded in their entirety. Consequently, Amy would watch the episode for 25 min, and then miss the (most important) ending, which significantly degrades her viewing experience and may drive her away from the mobile video streaming service. Using the quality-power adaptation framework will allow Amy to finish this episode at a slightly lower video quality, which in turns leads to a much better user experience.

7.2.6 Multistandards Mobile TV Testbed

In Section 3.6, we present the design and implementation of a complete, end-to-end, testbed for evaluating the *real* performance of mobile TV networks from various angles. The proposed testbed is totally open source and available to the research community. Thus, we believe it will stimulate and enable more research in improving the performance of mobile TV networks, which are expected to be pervasive in the near future. The testbed is developed for networks employing the popular DVB-H standard. The design of the proposed testbed is modular with well-defined interfaces between the hardware and software components. This enables updating different hardware/software components with minimal impacts on the others. Therefore, the testbed is useful for current and future generations of mobile video broadcast networks. In addition, because of its modularity, the testbed can easily be extended to support other broadcast standards, such as DVB-T (terrestrial), DVB-C (cable), DVB-S (satellite), and DVB-S/H (satellite to handhelds), among many others. This extension would enable studying the benefits/costs of integrating different standards in a comprehensive framework for providing digital video content to anybody, anywhere, and at anytime.

Bibliography

[1] Digital Video Broadcasting - Handheld (DVB-H) home page, 2009. `http://www.dvb-h.org/`.

[2] Mobile TV predicted to be a hit, 2007. `http://news.bbc.co.uk/2/hi/technology/6639249.stm`.

[3] X. Liang, B. Zhang, and R. Taylor. Development of the mobile phone television market in China. In *Proc. of Pacific Telecommunications Council (PTC'08)*, pages 1–22, Honolulu, HI, January 2008.

[4] Customers angered as iPhones overload AT&T, September 2009. `http://www.nytimes.com/2009/09/03/technology/companies/03att.html`.

[5] F. Wang, A. Ghosh, C. Sankaran, P. Fleming, F. Hsieh, and S. Benes. Mobile WiMAX systems: Performance and evolution. *IEEE Communications Magazine*, 46(10):41–49, October 2008.

[6] A. Ghosh, D. Wolter, J. Andrews, and R. Chen. Broadband wireless access with WiMAX/802.16: current performance benchmarks and future potential. *IEEE Communications Magazine*, 43(2):129–136, February 2005.

[7] S. Parkvall, E. Englund, M. Lundevall, and J. Torsner. Evolving 3G mobile systems: Broadband and broadcast services in WCDMA. *IEEE Communications Magazine*, 44(2):30–36, February 2006.

[8] Digital Video Broadcasting (DVB); transmission system for handheld terminals (DVB-H). European Telecommunications Standards Institute (ETSI) Standard EN 302 304 Ver. 1.1.1, November 2004.

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

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

[11] G. May. The IP Datacast system - overview and mobility aspects. In *Proc. of IEEE International Symposium on Consumer Electronics (ISCE'04)*, pages 509–514, Reading, UK, September 2004.

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

[13] 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.

[14] ATSC mobile DTV standard, 2009. `http://www.openmobilevideo.com/about-mobile-dtv/standards/`.

[15] D. Tran and T. Nguyen. Broadcasting techniques for video on demand in wireless networks. In B. Furht and S. Ahson, editors, *Handbook of Mobile Broadcasting: DVB-H, DMB, ISDB-T, and MediaFLO*, chapter 24, pages 675–696. Auerbach Publications, Boca Raton, FL, April 2008.

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

[17] K. Iizuka, H. Kawamura, T. Fujiwara, K. Kagoshima, S. Kawama, H. Kijima, M. Koutani, S. Toyoyama, and K. Sakuno. A 184 mW fully integrated DVB-H tuner with a linearized variable gain LNA and quadrature mixers using cross-coupled transconductor. *IEEE Journal of Solid-State Circuits*, 42(4):862–871, April 2007.

[18] Evolution of DVB-T front-end receivers through integration, June 2007. `http://www.dibcom.info/Images/Upload/pdf/whitePaper2_integration_MD_V2.pdf`.

[19] 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.

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

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

[22] DVB-H Paris mobile TV customer trial summary page. `http://www.dvb-h-online.com/Services/services-paris-canalplus.htm`.

[23] 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.

[24] M. Hefeeda and C. Hsu. On burst transmission scheduling in mobile TV broadcast networks. *IEEE/ACM Transactions on Networking*, July 2009. Accepted to appear.

[25] C. Hsu and M. Hefeeda. Time slicing in mobile TV broadcast networks with arbitrary channel bit rates. In *Proc. of IEEE INFOCOM'09*, pages 2231–2239, Rio de Janeiro, Brazil, April 2009.

[26] C. Hsu and M. Hefeeda. Broadcasting video streams encoded with arbitrary bit rates in energy-constrained mobile TV networks. *IEEE/ACM Transactions on Networking*, August 2009. Accepted to appear.

[27] T. Lakshman, A. Ortega, and A. Reibman. VBR video: Tradeoffs and potentials. *Proc. of the IEEE*, 86(5):952–973, May 1998.

[28] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network Magazine*, 17(6):6–16, November/December 2003.

[29] 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.

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

[31] C. Hsu and M. Hefeeda. On statistical multiplexing of variable-bit-rate video streams in mobile systems. In *Proc. of ACM Multimedia'09*, pages 411–420, Beijing, China, October 2009.

[32] V. Ollikainen and C. Peng. A handover approach to DVB-H services. In *Proc. of IEEE International Conference on Multimedia and Expo (ICME'06)*, pages 629–632, Toronto, Canada, July 2006.

[33] M. Rezaei, M. Hannuksela, and M. Gabbouj. Tune-in time reduction in video streaming over DVB-H. *IEEE Transactions on Broadcasting*, 53(1):320–328, March 2007.

[34] C. Hsu and M. Hefeeda. Bounding switching delay in mobile TV broadcast networks. In *Proc. of ACM/SPIE Multimedia Computing and Networking (MMCN'09)*, pages 72530A:1–72530A:12, San Jose, CA, January 2009.

[35] C. Hsu and M. Hefeeda. Using simulcast and scalable video coding to efficiently control channel switching delay in mobile TV broadcast networks. *ACM Transactions on Multimedia Computing, Communications, and Applications*, October 2009. Accepted to appear.

[36] C. Hsu and M. Hefeeda. Video broadcasting to heterogeneous mobile devices. In *Proc. of IFIP Networking'09*, pages 600–613, Aachen, Germany, May 2009.

[37] C. Hsu and M. Hefeeda. Multi-layer video broadcasting with low channel switching delays. In *Proc. of IEEE International Packet Video Workshop (PV'09)*, Seattle, WA, May 2009.

[38] F. Hartung, U. Horn, J. Huschke, M. Kampmann, T. Lohmar, and M. Lundevall. Delivery of broadcast services in 3G networks. *IEEE Transactions on Broadcasting*, 53(1):188–199, March 2007.

[39] C. Cicconetti, L. Lenzini, E. Mingozzi, and C. Eklund. Quality of service support in IEEE 802.16 networks. *IEEE Network Magazine*, 20(2):50–55, March 2006.

[40] 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.

[41] M. Takada and M. Saito. Transmission system for ISDB-T. *Proc. of the IEEE*, 94(1):251–256, January 2006.

[42] Radio broadcasting systems: Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers. European Telecommunications Standards Institute (ETSI) Standard EN 300 401 Ver. 1.3.3, May 2001.

[43] DVB-H global mobile TV : FAQ, 2008. `http://dvb-h.org/faq.htm`.

[44] FLO forum home page, 2008. `http://www.floforum.org/`.

[45] Digital Video Broadcasting (DVB) home page, 2008. `http://www.dvb.org/`.

[46] Digital Video Broadcasting (DVB); framing structure, channel coding and modulation for digital terrestrial television. European Telecommunications Standards Institute (ETSI) Standard EN 300 744 Ver. 1.5.1, June 2004.

[47] Digital Video Broadcasting (DVB); IP datacast over DVB-H: Set of specifications for phase 1. European Telecommunications Standards Institute (ETSI) Standard EN 102 468 Ver. 1.1.1, November 2007.

[48] S. Wenger, Y. Wang, and T. Schierl. Transport and signaling of SVC in IP networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1164–1173, September 2007.

[49] G. van der Auwera, P. David, and M. Reisslein. Traffic characteristics of H.264/AVC variable bit rate video. *IEEE Communications Magazine*, 46(11):164–174, November 2008.

[50] 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.

[51] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, September 2007.

[52] T. Wiegand, G. Sullivan, J. Reichel, H. Schwarz, and M. Wien. Joint draft 11 of SVC amendment. Technical Report JVT-X201, Joint Video Team, June 2007.

[53] Joint Video Team. Advanced video coding for generic audiovisual services. ITU-T Rec. H.264 & ISO/IEC 14496-10 AVC, March 2005.

[54] Video transmission for mobile wireless, scalable video broadcasting in 4G cellular deployments, 2009. `http://www.hhi.fraunhofer.de/fileadmin/hhi/downloads/BM/ICC2009/SVC-6c-low.pdf`.

[55] H.264 scalable video coding, what you need to know, 2009. `http://www.streaminglearningcenter.com/articles/60/1/`.

[56] Stretch Inc. announces support for H.264 scalable video codec. `http://www.stretchinc.com/news/pr_031008.php`.

[57] M. Tagliasacchi, G. Valenzise, and S. Tubaro. Minimum variance optimal rate allocation for multiplexed H.264/AVC bitstreams. *IEEE Transactions on Image Processing*, 17(7):1057–1143, July 2008.

[58] Z. He and D. Wu. Linear rate control and optimum statistical multiplexing for H.264 video broadcast. *IEEE Transactions on Multimedia*, 10(7):1237–1249, November 2008.

[59] M. Jacobs, J. Barbarien, S. Tondeur, R. Van de Walle, T. Paridaens, and P. Schelkens. Statistical multiplexing using SVC. In *Proc. of IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB'08)*, pages 1–6, Las Vegas, NV, March 2008.

[60] Statistical multiplexing over IP, 2008. `http://www.motorola.com/staticfiles/Business/Products/TV%20Video%20Distribution/Video%20Distribution/Multiplexers/SX-1000/_Documents/_Static%20files/SP_StatmuxIP.pdf`.

[61] Joint Video Team. Advanced video coding for generic audiovisual services. ITU-T Rec. H.264 & ISO/IEC 14496-10 AVC, November 2007.

[62] 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.

[63] Digital Video Broadcasting (DVB); IP datacast over DVB-H: Content delivery protocols (CDP) implementation guidelines. European Telecommunications Standards Institute (ETSI) Standard EN 102 591 Ver. 1.1.1, October 2007.

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

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

[66] Grass Valley Opal II mobile tv encapsulator, 2009. `http://www.grassvalley.com/docs/DataSheets/transmission/opal_ii/CDT-4009D-3.pdf`.

[67] P. Seeling, M. Reisslein, and B. Kulapala. Network performance evaluation using frame size and quality traces of single-layer and two-layer video: A tutorial. *IEEE Communications Surveys and Tutorials*, 6(2):58–78, July/September 2004.

[68] G. Sullivan and T. Wiegand. Rate-distortion optimization for video compression. *IEEE Signal Processing Magazine*, 15(6):74–90, November 1998.

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

[70] G. Creus and M. Kuulusa. *Mobile Phone Programming, Optimizing Mobile Software with Built-in Power Profiling*, chapter 25, pages 449–462. Springer Netherlands, 2007.

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

[72] E. Balaguer, F. Fitzek, O. Olsen, and M. Gade. Performance evaluation of power saving strategies for DVB-H services using adaptive MPE-FEC decoding. In *Proc. of IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'05)*, pages 2221–2226, Berlin, Germany, September 2005.

[73] Q. Zhang, F. Fitzek, and M. Katz. Cooperative power saving strategies for IP-services supported over DVB-H networks. In *Proc. of IEEE Wireless Communications and Networking Conference (WCNC'07)*, pages 4107–4111, Hong Kong, China, March 2007.

[74] M. Kornfeld. Optimizing the DVB-H time interleaving scheme on the link layer for high quality mobile broadcasting reception. In *Proc. of IEEE International Symposium on Consumer Electronics (ISCE'07)*, pages 1–6, Dallas, TX, June 2007.

[75] V. Vadakital, M. Hannuksela, and M. Gabbouj. Time-interleaved simulcast and redundant intra picture insertion for reducing tune-in delay in DVB-H. In *Proc. of IEEE International Packet Video Workshop (PV'07)*, pages 123–132, Lausanne, Switzerland, November 2007.

[76] M. Rezaei, M. Hannuksela, and M. Gabbouj. Video encoding and splicing for tune-in time reduction in IP datacasting (IPDC) over DVB-H. In *Proc. of IEEE International Conference on Multimedia and Expo (ICME'06)*, pages 601–604, Toronto, Canada, July 2006.

[77] M. Rezaei, M. Hannuksela, and M. Gabbouj. Video splicing and fuzzy rate control in IP multi-protocol encapsulator for tune-in time reduction in IP datacasting (IPDC) over DVB-H. In *Proc. of IEEE International Conference on Image Processing (ICIP'06)*, pages 3041–3044, Atlanta, GA, October 2006.

[78] T. Wiegand, G. Sullivan, G. Bjntegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003.

[79] E. Tan, L. Guo, S. Chen, and X. Zhang. PSM-throttling: Minimizing energy consumption for bulk data communications in WLANs. In *Proc. of IEEE International Conference on Network Protocols (ICNP'07)*, pages 123–132, Beijing, China, October 2007.

[80] Digital Video Broadcasting (DVB); IP datacast over DVB-H: Content delivery protocol. European Telecommunications Standards Institute (ETSI) Standard EN 102 472 Ver. 1.1.1, November 2006.

[81] Digital Video Broadcasting (DVB); DVB specification for data broadcasting. European Telecommunications Standards Institute (ETSI) Standard EN 301 192 Ver. 1.4.1, June 2004.

[82] M. Garey and D. Johnson. *Computers and Intractability: A Guide to The Theory of NP-completeness*. W. H. Freeman, 1979.

[83] P. Brucker. *Scheduling Algorithms*. Springer, 4th edition, 2004.

[84] Y. Bartal, S. Leonardi, G. Shallom, and R. Sitters. On the value of preemption in scheduling. In *Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'06)*, pages 39–48, Barcelona, Spain, August 2006.

[85] O. Braun and G. Schmidt. Parallel processor scheduling with limited number of preemptions. *SIAM Journal on Computing*, 32(3):671–680, 2003.

[86] R. Motwani, S. Phillips, and E. Torng. Nonclairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, August 1994.

[87] U. Schwiegeishohn. Preemptive weighted completion time scheduling of parallel jobs. In *Proc. of European Symposium on Algorithms (ESA'96)*, pages 39–51, Barcelona, Spain, September 1996.

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

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

[90] Spectrum LP49-DTV indoor antenna, 2008. `http://spectrum.co.kr/`.

[91] Enensys 1-watt RF power amplifier, 2008. `http://www.enensys.com/`.

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

[93] dvbSAM DVB-H solution for analysis, monitoring, and measurement, 2008. `http://www.decontis.com/`.

[94] WinTV-NOVA-T USB dvb-t receiver, 2008. `http://www.hauppauge.co.uk/`.

[95] Linux TV project, television with linux, 2008. `http://www.linuxtv.org/`.

[96] MAD-FLUTE project, 2008. `http://mad.cs.tut.fi/`.

[97] Joint Video Team. Joint scalable video model reference software. JSVM 14.0, May 2008.

[98] P. Lambert, W. de Neve, P. de Neve, I. Moerman, P. Demeester, and R. van de Walle. Rate-distortion performance of H.264/AVC compared to state-of-the-art video codecs. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(1):134–140, January 2006.

[99] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. Sullivan. Rate-constrained coder control and comparison of video coding standards. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):688–703, July 2003.

[100] 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.

[101] 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.

[102] 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.

[103] L. Wang and A. Vincent. Joint rate control for multi-program video coding. *IEEE Transactions on Consumer Electronics*, 42(3):300–305, August 1996.

[104] M. Rezaei, I. Bouazizi, and M. Gabbouj. Implementing statistical multiplexing in DVB-H. *International Journal of Digital Multimedia Broadcasting*, 2009, April 2009.

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

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

[107] M. Rezaei, I. Bouazizi, V. Vadakital, and M. Gabbouj. Optimal channel changing delay for mobile TV over DVB-H. In *Proc. of IEEE International Conference on Portable Information Devices (PORTABLE'07)*, pages 1–5, Orlando, FL, March 2007.

[108] S. Buchinger, S. Kriglstein, and H. Hlavacs. A comprehensive view on user studies: Survey and open issues for mobile TV. In *Proc. of European Interactive Television Conference (EuroITV'09)*, pages 179–188, Leuven, Belgium, June 2009.

[109] Y. Wang, J. Ostermann, and Y. Zhang. *Video Processing and Communications*. Prentice Hall, 1st edition, 2001.

[110] T. Schierl, T. Stockhammer, and T. Wiegand. Mobile video transmission using scalable video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1204–1217, September 2007.

[111] H. Haddadi, M. Rio, G. Iannaccone, A. Moore, and R. Mortier. Layered H.264 video transmission with hierarchical QAM. *Journal of Visual Communication and Image Representation*, 17(2):451–466, April 2006.

[112] C. Hellge, T. Schierl, and T. Wiegand. Mobile TV using scalable video coding and layer-aware forward error correction. In *Proc. of IEEE International Conference on Multimedia and Expo (ICME'08)*, pages 1177–1180, Hannover, Germany, April 2008.

[113] U. Ladebusch and C. Liss. Terrestrial DVB (DVB-T): A broadcast technology for stationary portable and mobile use. *Proc. of the IEEE*, 94(1):183–193, January 2006.

[114] B. Li and J. Liu. Multirate video multicast over the Internet: An overview. *IEEE Network Magazine*, 17(1):24–29, January/February 2003.

[115] A. Ganjam and H. Zhang. Internet multicast video delivery. *Proc. of the IEEE*, 93(1):159–170, January 2005.

[116] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proc. of ACM SIGCOMM'96*, pages 117–130, Palo Alto, CA, August 1996.

[117] K. Daoud. Performance comparison of the DVB-H and FLO mobile broadcasting systems. In *Proc. of IEEE International Symposium on Consumer Electronics (ISCE'07)*, pages 1–7, Dallas, TX, June 2007.

[118] Y. Liang. Real-time VBR video traffic prediction for dynamic bandwidth allocation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(1):32–47, February 2004.

[119] G. Gardikis. Mobile multimedia broadcasting standards: Technology and practice. In F. Luo, editor, *Transport and Time-Slicing Mechanisms in Multistandards for Mobile Broadcasting*, chapter 9, pages 281–293. Springer, November 2008.

[120] A. Bria. Cost-based resource management in hybrid cellular-broadcasting systems. In *Proc. of IEEE Vehicular Technology Conference (VTC'05-Spring)*, pages 3183–3187, Stockholm, Sweden, May 2005.

[121] V. Aggarwal, R. Caldebank, V. Gopalakrishnan, R. Jana, K. Ramakrishnan, and F. Yu. The effectiveness of intelligent scheduling for multicast video-on-demand. In *Proc. of ACM Multimedia'09*, pages 421–430, Beijing, China, October 2009.