University of Cincinnati

Date: 2/7/2011

I, Ngozi V Uti, hereby submit this original work as part of the requirements for the degree of Doctor of Philosophy in Computer Science & Engineering.

It is entitled:
Real-Time Mobile Video Compression and Streaming: Live Video from Mobile Devices over Cell Phone Networks

Student's name: Ngozi V Uti

This work and its defense approved by:

Committee chair: Yizong Cheng, PhD

Committee member: Richard Fox, PhD

Committee member: Fred Annexstein, PhD

Committee member: Raj Bhatnagar, PhD

Committee member: Dieter Schmidt, PhD
Real-Time Mobile Video Compression and Streaming: 
Live Video from Mobile Devices over Cell Phone Networks

By

Ngozi Victoria Uti

M.S. Computer Science, Indiana University, 2004
B.S. Computer Science, summa cum laude, Northern Kentucky University, 2002
utinv@mail.uc.edu

A dissertation submitted to the Graduate School
College of Engineering and Applied Science
University of Cincinnati

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in Computer Science and Engineering

Advisor and Committee Chair: Yizong Cheng, PhD

February, 2011
Abstract

The limited computing resources on mobile phones, the demands of real-time requirements, and the variable and error-prone nature of the bandwidth of cell phone networks make the task of streaming live video from cell phones very challenging. As such, computational simplicity and efficiency are a requirement for video encoders on mobile devices. This research presents core components of a mobile video compression algorithm that has been developed in this project to compress real-time video from cell phones. This work shows how the careful selection of video compression components can be used to strike a delicate balance between the computationally complex nature of video compression and the efficient utilization of the limited computing resources available on cell phones. Although optimality is never claimed, a method for compressing and streaming real-time video of 15 frames per second has been developed. The video encoder uses 5-3 wavelet transformation and a new subband aligned integer run-length encoding technique to compress video in real-time on mobile devices. The wavelet video encoder is adaptive, highly scalable, and can gracefully adjust video compression levels to match changing cell phone network bandwidth conditions.

Further, because of the variability of the bandwidth of cell phone networks, the efficient streaming of real-time video over cell phone networks requires the ability to adapt the quality and amount of video being streamed to the available bandwidth. This research shows that without such adaptability, video frames will be dropped. Experiments presented herein show that without an adaptive framework over 50% of the video frames can be dropped. In response to this challenge, this research implements an application layer framework for the control of real-
time streaming video originating from mobile devices to better utilize available bandwidth. The approach taken here aims to align the quality and transmission rate of live streaming video with the capabilities of cell phone networks.

Using decision making and feedback from the receiving video decoder, this real-time mobile streaming video framework is able to sense network conditions and effectively predict the available bandwidth. This adaptive framework utilizes the scalable wavelet video encoder for video compression. In conjunction with the wavelet video encoder on the mobile device, the framework adapts in real-time the video quality and video frames transmitted per second to achieve a near 100% delivery rate. This work provides a thorough description of this framework along with numerous experimental results. Presented is a detailed examination of the features of the adaptive framework and how they relate to cell phone network conditions, the video being streamed, and the mobile computing resources available on the mobile device.
To my wonderful husband, Jonathan, our beautiful children, William and Ojie, and my families who always believed in me
Acknowledgements

This work was supported in part by the National Science Foundation (NSF) Graduate Research Fellowship and the Ford Foundation Pre-Doctoral Fellowship. These external multi-year fellowships, along with the University of Cincinnati Graduate Scholarship, and support from the Computer Science Department, made my graduate experience a very smooth one and provided a stipend as well as funding for my tuition, research equipment, publications, and travels.

Thanks to Dr. Yizong Cheng for being the best advisor and committee chair I could have ever hoped for. My special thanks go to Dr. Richard Fox from Northern Kentucky University (NKU) for enthusiasm, guidance, encouragement, and support that span my University career from my undergraduate days to this PhD. My sincere gratitude also goes to Dr. Fred Annexstein, Dr. Raj Bhatnagar, and Dr. Dieter Schmidt for their time and dedication on my dissertation committee.

I thank Dr. Randall Bramley from Indiana University (IU) for his support and encouragement during my MS degree at IU and through this PhD degree. Thanks also to Dr. Herbert Dershem from Hope College for introducing me to research and publications during the NSF Research Experience for Undergraduate (REU) summer program and support on into graduate school. Special thanks to Professor Teresa Bright-Snodgrass from NKU for taking the time to stop in the hallway to recommend the NSF REU summer program to me, which led me to consider graduate education in the first place.
# Table of Contents

List of Figures ix

List of Tables xi

Chapter 1 Introduction 1

1.1 Statement of Problem 1

1.2 The Significance of Real-Time Mobile Streaming Video 3

1.3 The Complete Solution 6

1.4 Methodology for the Complete Solution 8

1.5 Structure of Dissertation 13

Chapter 2 Literature Review 14

2.1 Introduction 14

2.2 Related Works 14

2.3 Summary 18

Chapter 3 Challenges of Real-Time Video Compression and Streaming from Mobile Devices 21

3.1 The Concept of “Real-Time” for Video Streaming 22

3.2 The Limited and Variable Bandwidth of Cell Phone Networks 25

3.3 The Nature of Raw Video Frames from the Cameras of Mobile Devices 30

3.4 Limitations of Components of Video Compression Algorithms 36

3.5 Putting it all together: Strategy to Solve the Problem 41

Chapter 4 Addressing Video Compression Constraints 46

4.1 Introduction 47

4.2 Integer 5-3 Wavelet Transformation 48
4.3 Run-Length Encoding
   4.3.1 Full Horizontal Scan 52
   4.3.2 LL First Horizontal Scan 52
   4.3.3 Subband Aligned Scan 52
   4.3.4 Zig-zag Diagonal Scan 53
4.4 Experimental Setup 55
4.5 Experimental Results 58
4.6 More on SAIRLE 59
4.7 Summary 60

Chapter 5  MoStVid: Developing an Adaptive Framework for Streaming
Real-Time Video from Mobile Devices 61
5.1 Introduction to MoStVid 61
5.2 Real-Time Decision Making 62
5.3 Experimental Setup and Results: Testing the Performance of MoStVid 68
5.4 Summary 75

Chapter 6  Expanding MoStVid: Adapting Video Frame Rates and Data Sizes for Enhanced Video Quality 76
6.1 Introduction 76
6.2 Methodology on Mitigating Dropped Video Frames 77
6.3 Adaptive Wavelet Video Compression 81
   6.3.1 Quantization and Adaptive Video Qualities 82
6.4 MoStVid 87
   6.4.1 The MoStVid Framework 87
6.4.2 Decision Making Engine 90
6.4.3 Balancing Tradeoff between Video Compression and fps 95
6.4.4 Mobile-Network-Condition Awareness Features 97
6.4.5 Adaptations to Limited Mobile Computing Resources 99
6.5 Experimental Setup 102
6.6 Experimental Results 105
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7</td>
<td>Summary</td>
<td>115</td>
</tr>
<tr>
<td>7</td>
<td>Future Research Directions</td>
<td>117</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions</td>
<td>120</td>
</tr>
<tr>
<td>8.1</td>
<td>Contribution of Research</td>
<td>120</td>
</tr>
<tr>
<td>8.2</td>
<td>Concluding Remarks</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>123</td>
</tr>
<tr>
<td>A</td>
<td>Brief Biography</td>
<td>133</td>
</tr>
<tr>
<td>B</td>
<td>List of Publications</td>
<td>134</td>
</tr>
<tr>
<td>B.1</td>
<td>Invited Book Chapter from this Dissertation</td>
<td>134</td>
</tr>
<tr>
<td>B.2</td>
<td>Conference Proceedings from this Dissertation</td>
<td>134</td>
</tr>
<tr>
<td>B.3</td>
<td>Journal Article in Progress from this Dissertation</td>
<td>134</td>
</tr>
<tr>
<td>B.4</td>
<td>Other Publications</td>
<td>135</td>
</tr>
</tbody>
</table>
List of Figures

Fig. 3.1. Delivered fps rate of fixed size video frames 27
Fig. 3.2. Percentage of delivery for various sizes of video frames 28
Fig. 3.3. YUV 4:2:0 in planar storage method 32
Fig. 3.4. The time for generating video frames in 1 second 34
Fig. 3.5. Real-time streaming of video originating from mobile devices 43
Fig. 4.1. Multi-resolution structure of a 3-level wavelet transformed video frame 51
Fig. 4.2. Full horizontal scan path 53
Fig. 4.3. LL first horizontal scan path 54
Fig. 4.4. Subband aligned scan path 54
Fig. 4.5. Zig-zag diagonal scan path 55
Fig. 4.6. Experimental setup for comparing integer and bit-plane run-length coding 57
Fig. 4.7. Sample video frame 59
Fig. 4.8. A two-level 5-3 wavelet transformed copy of the sample video frame from Fig. 4.7 60
Fig. 5.1. Transmitted fps versus delivered fps 64
Fig. 5.2. Average video frame data size in bytes 65
Fig. 5.3. Improved percentage of video frames delivered over time 69
Fig. 5.4. Transmitted fps over the same time interval as Fig. 5.3 70
Fig. 5.5. Percent of frames delivered and cumulative percent of frames delivered 70
Fig. 5.6. Percent of frames delivered and cumulative percent of frames delivered 71
Fig. 5.7. The dynamic adjustment of fps in response to fluctuations in bandwidth 72
Fig. 6.1. Wavelet video encoder 81
| Fig. 6.2. | Frame #199 of Foreman video quantized at various $q$ values | 84 |
| Fig. 6.3. | Frame #1 of Stephan video quantized at various $q$ values | 85 |
| Fig. 6.4. | The MoStVid Framework | 89 |
| Fig. 6.5. | Tradeoff between fps and video compression level | 96 |
| Fig. 6.6. | Heavily fluctuating bandwidth video stream | 106 |
| Fig. 6.7. | Steady bandwidth video stream | 108 |
| Fig. 6.8. | Lightly fluctuating bandwidth video stream | 109 |
| Fig. 6.9. | Plummeting bandwidth video stream | 110 |
| Fig. 6.10. | Gradually decreasing bandwidth video stream | 111 |
| Fig. 6.11. | Gradually increasing bandwidth video stream | 112 |
# List of Tables

Table 3.1. Data sizes of video frames in various formats and resolutions 35

Table 4.1. Average data size of compressed 400x240x320 pixel live video frames with \( q = 21 \) 58

Table 4.2. Average data size of compressed 400x240x320 pixel live video frames with \( q = 30 \) 58

Table 5.1. Decision making criteria and actions for MoStVid 67

Table 5.2. Statistics of all frame data sizes in this experiment 73

Table 5.3. Data size statistics of video frames with 100% delivery success 73

Table 5.4. Video rate summary of the data sizes of video frames with 100% delivery success 74

Table 6.1. Summary of How Adjustments in \( q \) Affect the Resulting Video Quality 86

Table 6.2. MoStVid decision making rules 94

Table 6.3. MoStVid’s adaptation features for changing bandwidth and video patterns 99

Table 6.4. Profile of cell phone networks 103

Table 6.5. Summary of bandwidth patterns occurring in 60 experiments on networks 105

Table 6.6. Best case average values for \( q \), fps, and percentage of delivered frames 113

Table 6.7. Overall average values for \( q \), fps, and percentage of delivered frames 114

Table 6.8. Worst case average values for \( q \), fps, and percentage of delivered frames 114
Chapter 1  Introduction

1.1 Statement of Problem

This doctoral research focuses on the creation of efficient video compression and streaming algorithms to enable the real-time streaming of videos originating from mobile devices over low and variable bandwidth of cell phone networks. Mobile phones will become predominant and overtake PCs for World Wide Web access by 2013 [1]. Due to the proliferation of mobile communication devices, mobile phones and cell phone networks are rapidly becoming the de facto system of communication and even data transmission in many parts of the world. Just a few years ago in many developing regions of the world, access to a telephone was considered a luxury, but today mobile phones are a common aspect of everyday life. Privatized cell phone networks have succeeded in extending international communications access to regions where state phone monopolies had failed to provide even basic service. As an extreme example, in Somalia cell phone companies have extended communications services to the populace over the past decade, despite the collapse of the national government in 1992 [2].

Contemporary mobile phones have the potential to do much more than old style telephones since they are in reality light computing devices. Aside from the obvious use as telephones, they offer a number of other computational and telecommunication capabilities. In support of communication, for instance, people store address and phone books, calendars and notes on their
mobile phones. Thousands of other applications are available including the ability to access the World Wide Web. Additionally, many mobile phones are capable of producing and viewing high resolution photos and videos. It is only a matter of time before mobile phone users come to expect the ability to stream real-time video from mobile phones.

The advantages of streaming real-time video from mobile phones are explored in section 1.2 of this chapter. The timeliness of this topic is illustrated by the newest Apple iPhone [3], which can capture video and transmit it in real-time to other iPhones. Most mobile phones today only capture video, store the video in a data file, and then require that the file be transmitted over the cell phone network to a recipient to be played once the entire transmission has been completed, as opposed to a live video stream. The iPhone is able to accomplish the live streaming video only if it is transmitted over Wireless Fidelity (WiFi).

There are significant challenges that must be dealt with to realize the capability of transmitting live streaming video over cell phone networks. First is the inherently low and variable bandwidth of cell phone networks. Cell phone network performance is impacted by a number of independent problems with the potential for dropping numerous packets. Because the size of a raw video stream greatly exceeds the average bandwidth capacity of cell phone networks, the video must undergo some form of compression prior to transmission. The second challenge is that mobile phones and other mobile devices lack computing resources to easily compress raw video in real time. These devices have processors that compare with desktop/laptop processors of 5-10 years ago, limited memories, and battery management systems which further reduce computing power as the battery drains. The necessity of compressing raw video in real time for streaming requires video compression algorithms that can execute quickly
and yet the limited and variable bandwidth of the cell phone networks require that each video frame be reduced in data size to something that the cell phone network will not drop.

There are several video compression and streaming algorithms. These algorithms, designed for use on desktop/laptop computers, have computational demands that are not suited to the processors of mobile phones. Therefore, these algorithms cannot simply be ported to mobile phones. And so the ultimate challenge is to identify video compression and streaming techniques that can accomplish the task amid these difficult challenges while still delivering a video of watchable quality.

1.2 The Significance of Real-Time Mobile Streaming Video

The potential for using mobile devices and cell phone networks to send and receive real-time video is significant. Applications can extend well beyond the video phone idea of seeing who you are talking to. The ability of individuals to stream video in real time will open up new opportunities for improved emergency response, document human rights atrocities, aid law enforcement, provide a new tool for journalistic blogging, and improve education in areas that have poor educational capabilities, as well as the more obvious uses as a form of entertainment.

For medical applications, the availability of real-time video will allow first-responders (or even bystanders) to send detailed visual information regarding patient trauma to emergency physicians. This can greatly reduce the potential for miscommunication and misinformation when time is at a premium. Physicians could then provide critical feedback and guidance to first-responders in order to save lives. There are also significant applications in terms of data
collection for the support of tracking epidemic disease, particularly in places where computers are scarce. Similarly, emergency response for fire and utility interruptions can be greatly enhanced. Witnesses at the scene would be able to stream live video to responders in order to help guarantee appropriate response to developing situations.

A case in point is the Open Data Kit project implemented by Anokwa et al, wherein mobile phones were used in the Academic Model for the Prevention And Treatment of HIV program to collect data, including stored video, critical to the monitoring, prevention, and treatment of HIV, Malaria, and Tuberculosis in Kenya [4]. An example of using live streaming video is that of ReMoteCare, a system that combines streaming video with a wireless sensor network of pulse oximeters. This combined technology provides health monitoring for an aged care facility with the video streamed directly to a local computer rather than over a network [5]. The ability to efficiently stream real-time video from mobile devices would significantly enhance programs like these.

There are numerous possible law-enforcement applications. The delivery of real-time video already provides police and other security professionals with the ability to perform remote monitoring of a variety of settings. These include routine traffic (for instance, the monitoring of drivers who run red lights or for traffic control), monitoring of people in airports and train stations, crowd control, and even home owner monitoring. While these forms of streaming video are already in use, they either use a private network or the Internet. By streaming directly from a mobile device, surveillance can be performed more cheaply to enhance criminal investigations.

There are a number of economic and development applications as well. Mobile phones have already helped overcome the “information famine” faced by many small business operators in
developing countries and rural areas. For example in [6], Tanzanian farmers use mobile phone communications to exchange important information about the supply and demand of products between rural-based farmers and wholesalers in the commercial capital. The ability of these small entrepreneurs to send real-time visual information regarding products and services will help to further advance economic development.

Mobile phones are often used to capture events as they unfold. In the youth protests in Iran in 2009, much of the violence from government forces was captured and shared. However, without streaming video, the person capturing the video must make sure that there is time to transmit the file after it has been saved. If the person is caught and the mobile phone confiscated, the video can be erased without having ever been seen by others. Therefore, streaming live video can support journalists who are creating video blogs in real time as well as the ordinary citizen who wants to immediately share their video with others.

Finally, streaming real-time video over mobile wireless networks can have a broad impact on education, especially to people with limited literacy skills and limited or no computer Internet access. A survey by The Sloan Consortium found that in the fall of 2008, 4.6 million students in the United States had signed up for at least one Online course [7]. Online courses may be offered asynchronously or synchronously. However, the expense for synchronous distance education is typically far greater than asynchronous techniques due to the requirement of real-time audio/video delivery. The ability to stream video over mobile wireless networks can reduce the cost of synchronous distance education dramatically. Further, in developing countries where state resources are limited, there are great challenges in the ability to provide any educational services to rural populations. Real-time mobile video streaming offers a cheap mechanism by which to deliver a portable yet rich educational environment to end users.
1.3 The Complete Solution

Research on the compression and streaming of real-time video originating from mobile devices must develop an efficient solution that can effectively balance the complex relationship between the many interacting components and sets of constraints which are inherent to the system. These include:

1. Cell phone computational resource constraints [8]
   a. Limited computing power
   b. Small memories
   c. Battery life constraints
2. Real-time video compression constraints
   a. Large amount of raw video data to be compressed in a limited time
   b. Variability of resulting compressed video frame data sizes to be transmitted
   a. Packet delay and losses due to congestion and buffer over-flow in routers in the wired portion of the network
   b. Bit error packet corruption and loss in wireless links
   c. Packet losses and delay due to mobility handoff (transfer of an on-going call from one base station to another)

Solving the problem of using mobile devices to stream real-time video originating demands a triple-pronged solution which addresses each of these sets of constraints. First, the video compression algorithm must be computationally simple and must be able to reduce the amount of data to be transmitted to a size that can be streamed over cell phone networks. The video compression algorithm must be able to perform computations that do not overwhelm the limited computing resources on the mobile device. As such, the compression algorithm must possess the ability to produce highly compressed video using reasonably low computation so as not to
consume too much battery power. The algorithm must thus be elegant enough to achieve effective compression while not overburdening the small memory and the relatively weak CPU power found in mobile devices.

Secondly, the real-time constraints of live video demand that the video frames be transformed, compressed, and transmitted across the cell phone network in an efficient manner so that the receiving party perceives minimal latency. To meet this real-time requirement, the video compression algorithm must be able to fully compress each video frame within the time interval between a video frame and the next video frame. The compressed video frame should be transmitted into the network before the video camera on the device produces the next video frame.

Thirdly, the video compression algorithm must be adaptive, in that it needs to be able to adapt the quality and rate of the streaming video to the cell phone network bandwidth using the computing resources available at any given moment. The video compression algorithm must be able to adapt and perform well while streaming live video across error-prone wireless networks where available bandwidth can vary wildly due to external factors. Finally, the behavior of wireless network bandwidth which cause bandwidth variability must be understood and taken into account while streaming the video.

Because the successful delivery of real-time video originating from the cameras of cell phones demands balancing a number of discrete yet interacting constraints, an effective solution must examine the system as a whole. More specifically, the delivery of real-time video from mobile devices over wireless networks demands an adaptive framework which is multifaceted and which can dynamically adjust in response to the categories of constraints listed above: cell
phone computational constrains, real-time video compression constraints, and cell phone network bandwidth constraints.

It is important to stress that the discovery of the true nature of the constraints identified above was only achieved by undertaking experiments utilizing real mobile phones, real-time video, and real cell phone networks. A realistic solution to the challenge of streaming real-time video originating from mobile phones demands grounding in the real world of hardware and network constraints, because the situation is highly dynamic. By using real mobile phones and real cell phone networks this research was able to achieve a depth of insight into the true complexities of these interacting challenges which would not otherwise be possible. The following section discusses this unique methodology of doing research on actual mobile devices, using real-time video from the cameras of the mobile devices, and streaming the compressed video over real cell phone networks.

1.4 Methodology for the Complete Solution

The goal of this dissertation is to solve the problem of compressing and streaming real-time video originating from the cameras of actual mobile devices over actual cell phone networks. The idea is to study the entire system (mobile device, real-time video, and cell phone network) and use the experience gained from the compression and streaming of real-time video from the cameras of mobile devices over cell phone networks to design a comprehensive solution that takes into consideration the computational limitations of mobile devices, the real-time video compression constraints, and the limited and variable bandwidth of cell phone networks. To this end, extensive experiments were conducted to observe, identify, and respond to the limitations of
the mobile devices, the time-sensitive constraints on the real-time video, and the factors that influence the fluctuations in the bandwidth of cell phone networks. This all-embracing real-world approach allows for a model-less and simulation-less solution which provides a high degree of insight into the true complexity of the problem.

A major advantage of doing research on real mobile devices and real cell phone networks is that this approach reveals a number of challenges that are inherent to the system. This dissertation, being the first to address the problem of streaming video in real-time from mobile devices over actual cell phone networks, has no precedent. As such the methodology of using actual mobile devices, real-time video from the cameras of the mobile devices, and also real cell phone networks is a necessary approach to achieve a realistic solution, although very expensive.

The initial challenge involved the computational limitations of mobile devices. It was observed that mobile devices have limited memories with processors that are very poor at floating point operations. Thus, computational simplicity is necessary to reduce the time spent compressing the real-time video. The experimental data shows that an integer-only video compression algorithm produces a result that allows mobile devices to compress video in real-time. Chapter 4 describes the integer-only 5-3 wavelet transform and sub-band aligned integer run-length encoding real-time mobile video compression algorithm developed to meet the constraints of the low computational capacity of mobile devices.

Next, attempts to stream the already compressed video frames in real-time reveal that over 50% of the video frames could be dropped, and the drop rate depends on both the changing data sizes of the video frames and the variable quality of the bandwidth of the cell phone networks at the time of video streaming. From this observation it becomes apparent that a real-time adaptive
video streaming framework is necessary to dynamically adapt the video to match the available bandwidth of the cell phone network. To address the problem of adapting the real-time video, this research identifies the following parameters which can be adapted:

- The frames per second (fps) rate of the video camera.
- The video quality (data sizes of the video frames).

As such, in chapter 5 a real-time adaptive video streaming framework is presented to adapt the fps rate of the video camera in response to the amount of video frames being dropped in the cell phone network. The adaptation of the fps rate of the video camera results in improved video frame delivery, but because the data sizes of the compressed video frames are highly variable, the system was still limited in that the variable data sizes of the video frames continues to cause larger video frames to be dropped more frequently.

A comprehensive solution to address all parameters requires that the video quality must also adapt in conjunction with the fps rate. For the video quality to adapt in real-time, three new problems must be overcome. These problems are:

1. The framework must sense in real-time the available bandwidth of the cell phone network.
2. The framework must generate a target frame data size to match the sensed bandwidth.
3. The framework must ensure that the compressed video frames (compressed to match the target frame data size) do not result in unacceptable and unwatchable video quality.

From the above video quality requirements, a question which has never been addressed arose. Given a target frame data size, how does a mobile device with very limited computing
resources determine in real-time the video quality of the compressed video frame so that unacceptable video is not streamed? The commonly used peak signal-to-noise ratio (PSNR) metric cannot be used to determine real-time mobile video quality due to its computational intensity and the necessity to save the raw uncompressed video frames on the mobile device for use in the PSNR computations. Because the quality of the real-time video must be known in order to avoid streaming video frames of unacceptable quality, this dissertation developed a computation-free solution to determine video quality in real-time and also impose an upper and lower bound in the video compression levels of the adaptive video encoder (these details are presented in chapter 6).

From the above challenges, this research identifies three conditions that can occur while streaming any type of video:

a) The bandwidth of the cell phone network may be relatively steady, increasing, or decreasing.

b) The data size of the video frames may be relatively steady, increasing, or decreasing.

c) Given the upper and lower bounds of the video quality, the adaptive video encoder may be able to compress the video frame to be on target to match the target frame data size, above target, or below target.

There are 27 possible combinations of the above three conditions. The adaptive framework must adapt to each possible combination. These 27 possible combinations provide the basis from which to enhance the adaptive framework to handle all possible cases of fluctuations in the bandwidth, changes in the video frame data size, and whether the video frames can be compressed to match the target frame data size for the sensed bandwidth. From over 100 real-
time video streaming experiments from mobile phones and over various cell phone networks, with the goal of achieving the highest rate of video frame delivery, this research developed 27 rules with actions (Table 6.2) to dynamically adapt the fps rate in conjunction with the continuously generated target frame data size.

The fps adjustment actions associated with each of the 27 rules started off as educated guesses, and were fined-tuned to their final values over the course of more than 100 real-time video experiments over several cell phone networks. These 27 rules occurred repeatedly during the experiments, providing ample opportunity for each rule to be thoroughly tested and fine-tuned. After the fine-tuning experiments, the rules were fixed. Finally, an additional 60 experiments were performed to test the performance of the framework on various cell phone networks. These 60 new experiments using real-time video, reported in chapter 6, highlight the ability of the framework to deal with new and uncertain situations. The author believes that the generated target frame data size and the actions of the 27 rules of Table 6.2 are near-optimal as they allow the framework to be robust and achieve a near 100% video frame delivery rate, while adapting to the uncertainties of the bandwidth of cell phone networks observed in various bandwidth patterns.

It is worthy of note that the model-less and simulation-less approach of conducting this research on real mobile devices and over real cell phone networks allowed for the development of a realistic and complete solution to the problem of compressing and streaming real-time video originating from the cameras of mobile devices. This methodology makes this research unique.
CHAPTER 1 INTRODUCTION

1.5 Structure of Dissertation

This dissertation is organized as follows. Chapter 2 provides a literature review and describes works related to real-time mobile video compression and streaming. Chapter 3 identifies and analyzes the constraints and challenges of compressing and streaming real-time video originating from mobile devices. Beginning from chapter 4 a solution for the problem is presented. Chapter 4 addresses video compression constraints and develops a real-time video compression algorithm suitable for use on mobile devices. Chapter 5 presents MoStVid, an adaptive real-time video streaming framework for streaming video from mobile devices. Continuing in chapter 6, in addition to adapting the frame per second rates, the MoStVid framework is enhanced to include video quality adaptations. Chapter 6 also discusses in-depth the features of the MoStVid framework that are able to address and balance the challenges and constraints of streaming real-time video originating from mobile devices to achieve a near 100% video delivery rate. Future research directions and conclusions are presented in chapters 7 and 8.
Chapter 2  Literature Review

2.1 Introduction

At the time of this writing, the real-time compression and streaming of video originating from mobile devices over cell phone networks has received very little attention in the literature. In the mid 1990’s and early 2000’s researchers explored the potential of mobile video streaming. However, during this period mobile devices lacked the computational and memory resources necessary to do actual video streaming. Nonetheless, these early researchers identified some important challenges in the use of mobile devices to stream real-time video. The following section describes those works which addressed some of the constraints related to the streaming of real-time video over cell phone networks.

2.2 Related Works

In 1994, Belzer et. al [10] developed a hardware-wavelet-based adaptive video coding scheme for mobile wireless networks which examined some of the challenges of streaming video over low, variable, and error-prone bandwidth. Their adaptive video scheme transmitted simulated video over a simulated indoor wireless network. Interestingly, the bandwidth of the simulated network utilized was remarkably similar to current existing wireless network bandwidth. This
work is notable because it is one of the few studies that simulated conditions realistically. Their research took into account the hardware and power consumption considerations of video compression on mobile devices which do not have extensive computational capabilities.

In 1995, to address the issue of dropped video frames in networks, Kanakia et. al [11] proposed an adaptive scheme which uses feedback from a bottleneck switch in a simulated packet-switched network to regulate the bits per frame at the video encoder. Their results show that varying the bits per frame results in a graceful degradation of video quality and produces better picture quality than priority loss schemes where I and P MPEG video frames are given higher priority. However, their experiment was performed with higher quality video and with more bandwidth than is available on cell phone networks. Their control mechanism relies on congestion feedback information from a simulated bottleneck switch. Congestion feedback from network switches is not readily available to application layer video encoders. So while these results are useful, they cannot be implemented on real mobile devices or the public Internet. Because of the enormous scope of the public Internet, where packets can traverse any route, any such feedback from a particular congested switch is insignificant.

In 2002, Lee and Dey [12] evaluated the impact of the energy consumption of wavelet compression of images transmitted over wireless networks. This work’s primary concern is the elimination of some wavelet computations in order to reduce energy consumption for the wavelet transform component of image compression. They developed an energy efficient wavelet image transform algorithm that takes advantage of the numeric distribution of high pass wavelet coefficients to avoid the computation and transmission of certain high-pass wavelet coefficients. By avoiding the computation of some wavelet coefficients, the algorithm is able to save on
battery power consumption on the mobile device. The wireless bandwidth that will otherwise be needed for the transmission of the same coefficients is also saved.

Recent work has been done on the related but reverse problem of video transmission to mobile devices. The challenges and constraints when transmitting video to mobile devices include: battery life, memory size, cache size/bandwidth, CPU speed, display resolution, and bandwidth [13], [14]. Although related, the problem and challenges of transmitting video to mobile devices is distinct from the challenges of streaming video originating from mobile devices in real-time.

Researchers have taken two broad approaches to address the problem of adapting video transmissions to mobile devices. One approach is the implementation of an adaptation proxy server between a video server and a mobile device. The adaptation proxy server generally scales the video quality down and scales/crops the display size to match the bandwidth, memory, and display on the receiving mobile device. The second approach used is to have multiple versions/qualities of the same video saved on the video server. When a request for video is made from a mobile device, the video server selects a suitably low quality video for transmission to the mobile device.

To address the problem of transmitting video to cell phones, Chen and Li [13] developed a proxy framework for adapting video to be streamed from a video server to mobile devices. In addition, their research identified modifications that can be made to the parameters of the H.264 video standard to achieve a reduced computation video decoding that is suitable for mobile devices. Al-Turkistany et al [15] developed an adaptive thin-client model which uses fuzzy logic for dynamically sensing the bandwidth and adapting the transmission of remote application
screens (images) to mobile devices in order to facilitate the task of running remote server applications from mobile devices.

Several approaches have been proposed to help mitigate the impact of dropped and delayed video frames to improve the receiver’s perceived video quality. These solutions can be grouped into two broad categories: application layer and network solutions. For application layer solutions, it is now commonplace for multi-media applications to send time sensitive video using the User Datagram Protocol (UDP) instead of the Transmission Control Protocol (TCP) thereby avoiding the unacceptable delays which can result from TCP’s congestion control algorithm [9]. The video receiver can delay the start of video playback to mitigate the effect of the variability of packet delays so that the user can view a smoother video. For dropped video frames, the video receiver can request a retransmission of key frames or replay previously received video frames in place of missing frames.

Network solutions are more involved. Ke and Chilamkurti [16] propose a network layer content-aware marking scheme for providing differentiated service to packets according to the importance of their contained I/P/B MPEG video frames. Using the proposed Enhanced Token Bucket Three Color Marker (ETBTCM), I frames (intra mode) are given the highest priority because other frames are dependent on I frames. If an I frame is lost or corrupted, all frames in the same Group of Pictures (GoP) are also corrupted. P frames (prediction mode) are given the next level of priority because the effect of a lost or corrupt P frame will propagate to earlier B frames and to all following frames in the same GoP. B frames (double prediction mode) are assigned the lowest priority because a lost or corrupt B frame does not affect any other frames. The authors demonstrate that the ETBTCM produced better quality video than legacy packet makers by limiting the drop rate of I and P video frames.
To combat the problem of lost/corrupt video frames in the network, Chow and Ishii [17] use a Server Diversity Multiple Description Coding technique to enhance the quality of video transmitted over ad hoc wireless networks. Other researchers ([18], [19]) use Multiple State Coding technique to transmit video over several wireless routes to improve the received video quality on the assumption that it is highly unlikely for congestion to occur on all routes.

Another solution to mitigate the problem of the delay and loss of video frames in the MAC layer of wireless networks is the IEEE 802.11e standard [20]. The IEEE 802.11e is an extension to the base IEEE 802.11 standard to provide differentiated Quality of Services (QoS) for video and delay-sensitive multimedia in the MAC layer of wireless networks. Differentiated QoS is provided using four access categories (AC): AC_VO for audio, AC_VI for video, AC_BE for best-effort, and AC_BK for background traffic. Highest priority is given to AC_VO, followed by AC_VI, then AC_BE, and AC_BK gets the lowest priority. It should be noted that despite congestion control schemes in the network, if the video sender and other applications continue to send packets into a congested network, the network will continue to be congested.

2.3 Summary

This dissertation is the first academic project to address the problem of compressing and streaming real-time video originating from the cameras of real mobile phones and over real cell phone networks. This research is timely in that mobile phones are just now maturing to the point where they can undertake demanding tasks such as the real-time streaming of video. Indeed, mobile phones are now light computing devices. However, tasks such as video streaming can be computationally expensive and data intensive, so careful algorithm design considerations are
essential if the improving but still limited resources of these devices are to be used to maximum effect.

Realistic research needs to be conducted on real cell phone devices using real cell phone networks which do not necessarily behave like simulated networks with predetermined assumptions on the quality of bandwidth. The bandwidth behavior of real world cell phone networks are beyond the control of the researcher and are constrained by factors including the user’s physical location and proximity to the base station, environmental conditions, external interference, and the dynamic workload on the network [21].

To the best of the author’s knowledge there are no standard libraries or platforms for capturing real-time video from mobile devices. To conduct research that examines the compression and streaming of real-time video from cell phones, the researcher must generate almost from scratch the method for capturing the raw video frames and compressing the video before transmission. Even though mobile devices can capture video in existing video formats, such as H.263 [22] and H.264/AVC [23] (originally aimed at compressing video at low bit rates) these videos cannot be streamed live in these formats, they can only be saved on the device. As such this research developed a wavelet video compression algorithm instead of examining video compression schemes based on the older discrete cosine transform used in the H.263, H.264/AVC, and other MPEG family of video. The wavelet video compression algorithm will be discussed in chapter 4.

Video compression and streaming is perhaps the most computationally expensive and data intensive operation that can be tasked on both the mobile device and the cell phone network. In addition to real world network issues and the lack of mobile video capturing libraries, any video
compression algorithm implementation for mobile devices cannot simply use existing compression algorithms originally designed for desktop computers. A real-time video compression algorithm on mobile devices must be scaled down to operate within the constraints of the computing resources available on the device and the error-prone bandwidth of wireless network.

More so, extensive computations consume energy that is limited in the battery life of mobile devices. As such a researcher cannot simply port over computationally expensive desktop video compression algorithms to mobile phones without very careful modifications to the algorithm so as to make efficient use of the limited resources available on mobile phones. Also, the resulting video needs to be compressed and adapted to match the variable bandwidth of cell phones networks. In addition wireless network bandwidth cannot be used effectively if video compression standards such as MPEG-4, originally designed for wired networks, are directly ported on wireless networks because those standards assume a level of network stability found only in wired networks [24], [25].
Chapter 3  Challenges of Real-Time Video Compression and Streaming from Mobile Devices

Being the first academic research to fully address the specific topic of compressing and streaming real-time video originating from the cameras of real mobile devices over existing cell phone networks, this chapter identifies and thoroughly analyzes the constraints and challenges involved in attaining this goal. There are many factors that complicate this problem including the limited computational resources of mobile phones, the low and variable bandwidth of cell phone networks, and the need for video compression and streaming algorithms that can be supported by both the mobile phones and cell phone networks. This chapter examines the problems involved. It is important to understand the real-time constraints and challenges of compressing and streaming video from mobile devices in order to design efficient video compression and streaming techniques that are able to work within the limited computational resources and bandwidth available to mobile devices.

This chapter is organized as follows. Section 3.1 clarifies the notion of “real-time” for video streaming. Section 3.2 will examine the limitations imposed on this problem by the bandwidth of cell phone networks. Section 3.3 considers the nature of the video that mobile phones can produce. Section 3.4 identifies the limitations of components of video compression algorithms.
with respect to the constraints and challenges of real-time mobile video. Finally, section 3.5 combines all of the concepts from this chapter into a complete strategy to solve the problem.

3.1 The Concept of “Real–Time” for Video Streaming

The concept of “real-time” for streaming video is a complex and at times ambiguous issue. There is neither a single clear definition of the term nor a scholarly consensus on what does or does not qualify as real time. Kehtarnavaz and Gamadia [26] have examined the literature on this subject, and have provided a very useful three-part categorization of the different views of what denotes real-time as tied to different applications and disciplines. In so doing, they present three different interpretations of what constitutes real-time video. These are real-time in a perceptual sense, real-time in a software engineering sense, and real-time in a signal processing sense. Each of these interpretations is based upon a certain degree of tolerance for computational delay in the processing and streaming of video. This section discusses and provides examples of each of these three definitions of real-time. This section will also provide an explanation for the utilization of real-time from a signal processing perspective as used in the remainder of this dissertation.

Real time in the perceptual sense relies first and foremost on the human perception of the video. If there is no discernable lag in the reception of the video as interpreted by the user, then the video qualifies as real-time. Crucial here is the ability of the human mind to string together images into a perceived smooth motion. Thus, even if frame rates are lower than optimal – research shows that humans can detect flickers at video frame rates below 48 frames per second (fps) [27] – users may still interpret the video as adequate at frame rates as low as 7 fps. This is
especially true given the expectations of lower-quality video available in resource-constrained environments such as mobile devices and low-bandwidth Internet connections. Nonetheless, there is a limit to what humans will tolerate, depending on the amount of motion in the video, the smoothness of the frame rates, and whether or not there are gaps evident in the reception of the video. Some researchers, including [28], have even described the delivery of slightly delayed video as real-time in this perceptual sense, so long as the recipient does not detect the delay resulting from processing a few future video frame samples. Thus, even if the future frames are slightly delayed while the current frame is being processed, the result to the user may still appear to be real-time video.

Real-time in the software engineering sense is concerned with how quickly an accurate output of a process can be made available in an expected amount of time. That is, real-time is achieved when a successful and predictable performance is made within a time frame that is required by an application [29]. Notably, from the software engineering point of view, video processing need not be completed as quickly or often as possible, but rather the output of the software process simply needs to be available on time, perhaps for input into another dependent software process. As an example, in [30], a quality control vision inspection tool using a camera to inspect products in an industrial environment at one second intervals could easily achieve real-time processing given that the inspection process is completed in 0.1 second(s), allowing ample time before the next input requiring inspection occurs. The camera might be capable of far more fps, but this would be unnecessary given that the application parameters only demand product inspection once per second. There is certain flexibility in this definition of real-time: as long as video processing is able to complete the particular task of video analysis in a predictable time, the video can be considered to be real-time.
In signal processing, video processing is considered to be real-time if each video frame is able to be compressed and streamed within the time frame between the production of two consecutive video frames. For example, for a 15 fps video camera, assuming the time to produce the video frame is negligible, there is 0.067 s between consecutive frames. A real-time video compression algorithm must be able to completely compress, package and stream each video frame into the network within 0.067 s, in time before the video camera generates the next video frame. As such, according to Kehtarnavaz [31], to be real-time, the time to execute the number of instructions in the video compression algorithm and video streaming process must not exceed the time between video frame samples. The definition of real-time in the signal processing sense is strict in that the mobile device must implement an efficient video compression algorithm that is able to utilize the limited computing resources on the device to compress the video frame fast enough to have it streamed without the computing resources on the device becoming a bottleneck. There are other factors that impact this restriction. The computing power available to a mobile device varies according to the battery resources available to the device at that time [15]. The size of the video frame will vary based on the complexity of its picture content. Larger sized video frames (in terms of data) may take longer to encode. If the bandwidth of the cell phone network is low, the compressed video frame will take longer to be streamed. The bandwidth of cell phone networks is highly variable. Therefore, to commit to real-time in the signal processing sense requires that all of these factors be taken into account.

For the remainder of this dissertation, the definition of real-time in the signal processing sense will be used when describing streaming video in real-time. This is because mobile devices typically lack the computing resources necessary for the accumulation and processing of several video frames in memory so as to have the video still appear to be real-time. Given these limited
computing resources, it is necessary for the mobile device to process each video frame as quickly as possible to free up the memory and CPU for use by the video camera to capture the next video frame to be compressed and streamed. In addition, the specific purpose of the video may not be known in advance. With a clear understanding of the impact of real-time requirements in the compression and streaming of live video from a signal processing sense, the following sections discuss the bandwidth of cell phone networks, and the nature of the video to be compressed and streamed in real-time.

3.2 The Limited and Variable Bandwidth of Cell Phone Networks

The limited and variable bandwidth of cell phone networks impose significant constraints upon the streaming of real-time video. Although 3G cell phone networks are rated to support upwards of 15 Mbps, the typical bandwidth available can be much less, for instance below 300 Kbps [32]. Among the more commonly used networks in the United States are the Enhanced Data rates for GSM Evolution (EDGE), High Speed Packet Access (HSPA) and Evolution-Data Optimized (EVDO) networks. The 2.5G EDGE network has a maximum rated bandwidth of 120 Kbps, but typical rates are around 30 Kbps [32]. The HSPA and EVDO are 3G networks with maximum rated bandwidth of up to 14 Mbps and 3.1 Mbps respectively [32]. 3G network coverage and bandwidth can be inconsistent even in major cities, with actual measured throughput typically ranging between 50 Kbps and 300 Kbps for HSPA, and about 90 Kbps for EVDO [32].

The causes of the degradation in performance from the rated bandwidth are manifold but include network congestion, bit errors in wireless links, and mobility handoff [9]. In addition,
the bandwidths of wireless links fluctuate significantly due to channel fading and interference [33]. The user’s physical location, mobility and environmental conditions all play a role in the available bandwidth of wireless networks [21]. Together, these various factors can result in highly variable and unpredictable bandwidth. It is critical that real-time video be adapted to meet the actual bandwidth available at the time of the video streaming.

The choice of a transport protocol is also necessary to insure that video is delivered in real time. There are two transport protocols available, User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). TCP is typically utilized for services which require a guarantee of delivery. Examples include FTP, e-mails, and Internet applications that are not time or delay sensitive. However, the time sensitive nature of real-time video requires the use of UDP in order to avoid delays in the delivery of packets. The unlimited retransmission of packets in TCP renders this protocol unacceptable for real-time video streaming. UDP does not guarantee delivery, but can deliver packets with minimal delay to meet the needs of real-time video streaming.

To illustrate the bandwidth variability in a real 3G HSPA cell phone network, experiments were conducted using UDP to stream 5 minutes of fixed size video frames from a mobile device over this 3G network. While the data sizes of actual video frames vary significantly, the use of fixed size video frames in this experiment highlights the influence of bandwidth on video delivery. Six different streams of video frames were sent, with data sizes ranging from 3 KB to 8 KB. The video frames were transmitted at a rate of 15 fps. Fig. 3.1 presents a graph of the delivered fps rate of each video frame data size. Notice the variability in the delivery rate. Smaller size video frames were delivered far more consistently and with far fewer dropped video
frames than larger size video frames. For example, at the 3 KB size an average of 14.2 fps were delivered, while at 8 KB only an average of 5.25 fps were delivered.

Fig. 3.2 shows the percentage of delivery of the video frames data sizes from the same experiment as Fig 3.1. The percentage of delivery drops with increasing frame data size. At the time of the experiments, video frames of 3 KB had a 94.6% delivery and video frames of 8 KB were delivered at only 35%, dropping nearly two of every three video frames. Clearly, at the
time of the experiments, the 3G network could not handle streaming video of 4KB video frames or larger.

There is an additional and potentially greater problem than the low delivery rate of large video frames. Some video compression algorithms use inter-frame compression techniques that create dependencies between multiple frames. With such an algorithm, a lost reference frame can exacerbate the problem of limited bandwidth leading to errors propagating across all dependent frames. In [33] an MPEG-4 video experiment utilized inter-frame compression with Intra (I) frames of approximately 10 KB and Bi-predicted (B) and Predicted (P) frames of approximately 4 KB. Based upon the above 3G bandwidth experiment, if I frames of size 10 KB were transmitted at the time of the experiment, their percentage of delivery would likely be less than 35% since frames of 8 KB were delivered at only 35%. These losses would propagate
across all dependent B and P frames, which would render these follow-up frames erroneous because they were received without their reference I frame. Indeed in that MPEG-4 experiment, each I frame had 15 dependent B and P frames in the same *Group of Pictures (GoP)*, all of which would be lost along with their reference I frame.

It should be noted that some mobile phone users are now using their phones to connect to WiFi hot spots. In such cases, their bandwidth is greatly improved as the hot spots typically connect to the Internet through broadband allowing for a much greater and more reliable bandwidth than the cell phone networks. While it may be simpler to stream video from a mobile phone via WiFi, this is presently an incomplete solution to the general problem discussed in this dissertation because WiFi has greatly reduced coverage areas compared to that of cell phone network coverage. Using WiFi also restricts the degree of roaming. A truly mobile person using their mobile phone will quickly go beyond the reception area for the WiFi that they are using and will have to locate a new hot spot. Since finding and establishing a new WiFi connection takes some time (possibly several seconds or more), any live streaming video will be interrupted.

There is progress being made in the availability of WiFi, particularly in metropolitan areas that either have poor cell phone network connectivity or extremely high traffic. As noted in [34], experiments in the Pittsburgh metropolitan area indicate that WiFi access is nearly as prevalent as 3G cell phone network coverage and that both offer roughly equivalent bandwidth when roaming is performed *slowly*. However, at present because WiFi coverage is less than cell phone network coverage, any streaming video technology that relies on WiFi will not provide the flexibility needed for the majority of mobile phone users. This is particularly true in developing regions of the world where WiFi coverage is very limited or non-existent.
3.3 The Nature of Raw Video Frames from the Cameras of Mobile Devices

The video software of mobile phones produce already compressed video which cannot be streamed from the device in real time. Further, the video software on these devices may not provide complete information regarding the resolution of video which can be obtained from the camera. This chapter deals with issues related to the compressing of raw video directly from the camera for the purpose of streaming it in real time. The information in this section, describing the nature of the raw video frames from mobile devices, is based upon the author’s direct examination of video cameras on a variety of mobile phones. The raw video frame formats and video frame resolutions described in this section cannot be obtained by simply viewing the mobile devices or their installed video software. Rather, this crucial information was obtained by programmatic communication with the video cameras in the mobile phones.

The nature of the raw video produced by mobile device cameras presents challenges for real time compression and streaming. The video cameras on mobiles device produce enormous amounts of data in each raw video frame. This is especially important given that these devices have limited memory, battery life, and CPU power, and yet the video must be compressed significantly. From the programmatic access to the cameras of mobile devices, it was observed that these devices can produce raw video frames in one of two color spaces, RGB and YUV. In the RGB color space, each pixel in a video frame is represented with three components red, green and blue. In the YUV color space each pixel is represented with a luminance Y, which is the grayscale component, and two chrominance components, U for hue and V for intensity.
For the RGB color space, the R, G, and B components of a pixel are typically stored in 24 bits, 8 bits per component. The mobile device cameras that produce video in the RGB color space can usually also produce sub-sampled RGB video where each component is stored with less than 8 bits per component. One such format stores each of the R, G and B components in 5 bits for a total of 15 bits per pixel. A slight variant uses 6 bits for the G component, thus storing each pixel in 16 bits. However, due to fewer bits being used to represent each component, these sub-sampled RGB formats typically produce video frames that are of poorer visual quality (even before video compression is applied). As a result, it is better to use the 24-bit RGB format and then apply forms of data reduction to best retain video quality before compressing the video.

YUV video can be sampled at 8-bits per Y, U, V component for a total of 24 bits per pixel. This is known as the YUV 4:4:4 format. Like RGB, the YUV video format also permits various sub-sampling formats. For mobile devices, the prevalent YUV format is YUV 4:2:0 where the Y component is stored in 8-bits, and the U and V components are sub-sampled and shared across 4 pixels in a block of two pixels high and two pixels wide (2x2 pixels). The rationale behind this approach is that the human eye is more sensitive to information loss in the luminance component and less sensitive to loss of information in the chrominance components. Since the U and V chrominance components are shared across 4 pixels, only 16 bits are used to represent the U and V component for each set of 4 pixels in the YUV 4:2:0 format. As such, each pixel is stored in only 12 bits, compared to the 24 bits required to store a single pixel in either of the RGB 24 or YUV 4:4:4 formats. Such a significant reduction in data size is very important when streaming video from mobile devices where computational and memory capacity are limited. Additionally, the YUV 4:2:0 format does not introduce a visible loss of quality unlike sub-sampled RGB formats.
YUV components may also be stored in either planar or packed methods. In planar storage, the Y components are stored in the first part of an array, and are followed by the U components, and the V components, while in packed storage the YUV components are stored pixel-wise. The planar method is commonly used for the storage of the YUV 4:2:0 raw video frames from the cameras of mobile devices. Fig. 3.3 illustrates an example of an 8x4 video frame with YUV 4:2:0 components stored in the planar method. Each 2x2 block of pixels has individual Y component values, but share the same U and V components.

The RGB color space is used primarily for representing pixels for display on standard desktop and laptop computer screens. However, most video compression algorithms have been optimized to exploit the spatial redundancy of video frames in the YUV color space. It is therefore worthwhile to convert from an RGB to YUV video format prior to compression in

![Diagram](image.png)

**Fig. 3.3.** YUV 4:2:0 in planar storage method. Each pixel in a block of 2x2 pixels in a video frame has an individual Y component but shares U and V components with other pixels in the block. A video frame of resolution 8x4 will require 48 bytes. The Y, U, and V components use 8 bits each.
order to decorrelate the RGB components and take advantage of the superior compression that can be achieved in the YUV color space. A conversion from 24-bit RGB to 12-bit YUV 4:2:0 video can be done with integer computations and shifts [35]. Thus, the conversion is quick and there is a 50% reduction in data size without visible loss of quality. Video compression can now be applied to further reduce the data size.

Aside from the format of the video frame, the resolution of the video frames produced is also a concern. Each video frame has a resolution measured in the number of horizontal pixels by the number of vertical pixels. Mobile phones can produce video frames in several different resolutions. The most common resolutions include: 120x120, 176x144, 176x176, 240x320, 320x240, 400x240, 352x288, and 640x480. Higher frame resolutions, however, come at a cost. As the resolution of a video frame increases, the camera on the mobile device cannot produce as many fps. For example, a video camera on a mobile device that is capable of producing video frames of 176x144 resolution at 15 fps may only be able to produce frames of 352x288 resolution at 7 fps, and 640x480 resolution at 3 fps. The reason for this is that the video camera on the mobile device requires more computational resources (CPU and memory) for larger dimensions, and hence, more time to generate the larger video frames.

The issue is that the data sizes of video frames grow at quadratic rate with increasing resolution. To double the apparent resolution of the video frame will result in a quadrupling of the data size. As the data size of the video frames increases, there is a similar quadratic growth in the computational cost of compressing the frames. A video frame with 176x144 resolution contains 25,344 pixels, while a video frame of 352x288 resolution contains 101,376 pixels and will require four times as much computation for the video compression algorithm. As such, it becomes considerably more difficult for the video compression algorithm to completely
Compress these larger video frames in the time available between frames. Fig. 3.4 shows how higher resolution frames can leave less time between frames for video compression in real time.

What happens if the video compression algorithm is not able to completely compress a video frame between the generation of two consecutive video frames? The result is that the camera will not generate the next video frame because the processor and memory necessary to do so are not available. There are simply not enough available computational resources to support in parallel the camera generating its next frame and the compression algorithm compressing the previous frame. The impact is that the computational resources of the mobile device coupled with the video compression algorithm become a bottleneck and effectively reduce the fps rate of the mobile device. Since existing video compression algorithms originally designed for desktop computers are computationally intensive, careful algorithm design must be employed to scale

**Fig. 3.4.** The time for generating video frames in 1 second. The computational cost of compressing video frames increase at a quadratic rate while the time in between video frames do not increase at a quadratic rate. (a) 352x288 resolution video frames generated at 7 fps produces 101,376 pixels and 1,216,512 bits per video frame in the YUV 4:2:0 format. (b) 176x144 resolution video frames generated at 15 fps produces 25,344 pixels and 304,128 bits per video frame in the YUV 4:2:0 format.
down video compression algorithms to be able to operate in real-time within the computational constraints of mobile devices.

Although the YUV 4:2:0 video format can significantly reduce the data sizes of raw video frames, the uncompressed videos remain very large. Regardless of the color space of the raw video, the amount of data in each raw video frame is such that they must be compressed before the video can be streamed. Table 3.1 shows data sizes of video frames using different mobile video resolutions and formats. The data size of the video frames can be obtained by multiplying the number of pixels in a video frame by the number of bits used to store each pixel. Higher resolution video can easily overwhelm the resources available for real-time video compression and streaming from mobile devices.

<table>
<thead>
<tr>
<th>Pixels in video frame</th>
<th>RGB 24 / YUV 4:4:4 (data size in bits)</th>
<th>YUV 4:2:0 (data size in bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>120x120</td>
<td>14,400</td>
<td>345,600</td>
</tr>
<tr>
<td>176x144</td>
<td>25,344</td>
<td>608,256</td>
</tr>
<tr>
<td>176x176</td>
<td>30,976</td>
<td>743,424</td>
</tr>
<tr>
<td>240x320</td>
<td>76,800</td>
<td>1,843,200</td>
</tr>
<tr>
<td>320x240</td>
<td>76,800</td>
<td>1,843,200</td>
</tr>
<tr>
<td>400x240</td>
<td>96,000</td>
<td>2,304,000</td>
</tr>
<tr>
<td>352x288</td>
<td>101,376</td>
<td>2,433,024</td>
</tr>
<tr>
<td>640x480</td>
<td>307,200</td>
<td>7,372,800</td>
</tr>
</tbody>
</table>
To better illustrate the need for video compression, note that a 5 minute 176x144 video in YUV 4:2:0 format at 15 fps will require 163 MB of bandwidth if uncompressed.

\[
5 \text{ min} \times \frac{60 \text{ sec}}{\text{min}} \times \frac{15 \text{ frames}}{\text{sec}} \times \frac{(176 \times 144) \text{ pixels}}{\text{frame}} \times \frac{12 \text{ bits}}{\text{pixel}} \times \frac{1 \text{ byte}}{8 \text{ bits}} \approx 163.15 \text{ MB}
\]

Clearly, given the large data sizes of uncompressed video at even a low resolution (176x144), a very efficient video compression algorithm is necessary for compressing the video to a data size that will be suitable for real-time streaming over cell phone networks.

### 3.4 Limitations of Components of Video Compression Algorithms

The discussion in this section will be limited to those aspects of video compression techniques that relate to the real-time video streaming challenges described throughout this dissertation. The major types of video compression techniques are based on either Discrete Cosine Transform (DCT) or Discrete Wavelet Transform (DWT). Both transforms have been adopted for use in standard image and video compression techniques. The DCT-based MPEG-4, and specifically part 10 which is also known as H.264/AVC, is the latest and most widely used standard video compression technique [36]. DWT is the image compression technique used in JPEG2000 [37]. Building on the JPEG2000 image compression, Motion JPEG2000 is a commonly used video compression technique where each frame in the video is individually compressed using the JPEG2000 image compression. Each of these video compression techniques has their merits in terms of computational cost, adaptability, video quality, data sizes, and bandwidth usage.
There are several profiles in the MPEG-4 standard. The constrained baseline profile and the baseline profile are the least complex MPEG-4 profiles that can be used for videoconferencing and mobile applications. The MPEG-4 standard, however, only defines the video decoder (recipient of a streamed video signal) and not the encoder (which performs the compression prior to streaming). A solution to stream real-time video from a mobile device using any of the MPEG-4 profiles must develop an encoder that will run on the mobile device to match the decoder defined in the standard. On the one hand, the integer forward DCT of the MPEG-4 encoder can be implemented with additions and shifts and without the use of a multiplier [38]. The integer operations are easy to implement without any loss in precision. This is a very important consideration given the low computational capacity of mobile processors. On the other hand the motion estimation component of MPEG-4 is the most computationally intensive step of the encoder [38], and can place excessive computational demands on mobile devices. The high computational complexity of MPEG-4 encoders is a significant reason for why real-time MPEG-4 encoders are not commonly available for mobile devices. However, the baseline profile of MPEG-4 includes three error resilient tools (flexible macroblock ordering, arbitrary slice order, and redundant slices) to deal with the high bit error rate that can occur in cell phone networks. The development of low-complexity MPEG-4 encoders that can operate in real-time on mobile devices and stream video over cell phone networks is a very challenging topic of research.

The MPEG-4 standard utilizes inter-frame compression techniques to reduce temporal redundancy across multiple video frames. There are four types of MPEG-4 video frames. The I frame is encoded independent of other frames. The P frame is encoded as the difference between the preceding or following I or P frame and the current frame. The B frame is encoded to rely on
both the previous and following I and P frames. The Bi-predicted Reference (BR) frame is a B frame that is used as a reference frame for preceding or subsequent frames. To reduce the complexity of the encoders, the B and BR frames are not allowed in the constrained baseline and baseline profiles. A GoP is a sequence of frames starting with an I frame and its dependent P, and B frames. In a GoP, the I frame, which is encoded with no prediction information, is larger than P and B frames. Overall, the temporal redundancy reduced across multiple video frames results in increased video compression in a GoP.

There are two consequences of GoP with respect to the real-time streaming of video over wireless links. First, the very important I frame, which is usually larger and contains many packets, can be lost or corrupted in the cell phone network. A lost or corrupt I frame will result in error propagation to all dependent frames in the GoP. This will intensify the effect of the already low bandwidth as even successfully received dependent frames will become erroneous. The second implication of GoP is that of adaptability. In streaming real-time video from mobile devices, it is desirable that the video be adaptable to better utilize the variable bandwidth of cell phone networks. MPEG-4 encoders typically allow adaptability on the GoP level but due to the inter-frame dependencies, individual video frames in a GoP are not adjustable [33]. The loss of frame-level adaptability makes it difficult to finely adapt an MPEG-4 video to match the available bandwidth. Also, the adjustment of the data size of a GoP to match a target size is usually done by the iterative adjustment of the quantization level of each video frame in the GoP [33]. The additional computational cost that can be incurred in the repeated quantization of video frames does not cohere with the demands of real-time video streaming from mobile devices.
The adaptability of the MPEG-4 standard continues to be improved. The *Scalable Video Coding (SVC)* is an extension of the MPEG-4 standard to provide adaptive video coding that allows the encoder to adapt various aspects of the video. For mobile applications, the scalable baseline profile of the SVC provides temporal, spatial, quality, and combined scalable coding for the video. Each scalable coding option provides adaptability by utilizing subsets of packets from the larger video stream. The adaptation of MPEG-4 video is an active area of research. See [39], [40]. Researchers have proposed schemes to improve the quality of MPEG-4 video and particularly to give priority to I frames that are being streamed in the wireless networks [16]. More information on H.264 and MPEG-4 can be found in [41].

The lightweight daViKo [42] is an example of an H.264/AVC commercial software implementation that can operate on mobile devices. The software is a scaled-down version of the highly optimized commercial daViKo desktop video conferencing tool. The lightweight daViKo software, however, only supports mobile peer-to-peer video conferencing over 802.11 WLAN/WiFi [43]. Schmidt, et al., do not report streaming video over cell phone networks. As a commercial product, the software is not open source.

The JPEG2000 image compression technique provides progressive transmission in four dimensions (quality, resolution, spatial location, and component) to support image improvement as more bits are received [44]. This technique allows the image quality and features to be continuously improved as more data is received. In the quality progressivity, the quality of the image improves with more data being decoded. For the resolution progressivity, the resolution of the decoded image grows by a factor of two from a thumbnail to a full size image. The spatial location progressivity allows an image to be transmitted using a *stripe* or *scan-based* approach, and is targeted for use by low memory printers and scanners. Component progressivity allows
an image to be transmitted such that various components (grayscale, color, text) of the image can be decoded as more bits are received. Because it offers a variety of choices of progressive transmissions, the JPEG2000 offers a large degree of flexibility in terms of image adaptations.

As with MPEG-4, the JPEG2000 standard only specifies the decoder. The encoder must be implemented to comply with the decoder in the standard. JPEG2000 encoders can be computationally intensive for mobile devices depending on the particular wavelet filter selected for video compression. Wavelet filters have various lengths and can be implemented using floating point or integer operations. The 5-3 and 9-7 wavelet filters are the two choices available in the JPEG2000 standard. The 5-3 wavelet filter is short and can be implemented using integer operations while the 9-7 is longer and requires floating point operations [44]. Although longer wavelet filters produce better compression, shorter integer wavelet filters are less computational intensive and are more suited to the computing resources on mobile devices. The Motion JPEG2000 video does not utilize inter-frame compression or motion estimation techniques. More information about JPEG2000 and Motion JPEG2000 can be found in [8], [45].

At a fundamental level, a major advantage of the DWT-based compression techniques over the DCT-based compression techniques is that image quality is much improved because the blocking artifacts that can result from DCT are absent in DWT images [46]. Generally, the image quality of DWT degrades more gracefully than that of DCT. DWT produces better quality images at very low bit rates. Instead of blocking artifacts, increased blurring is a side-effect of highly compressed DWT images. An exception can be found in the case of JPEG2000 where the DWT can be applied to small tiles of the image and can result in blocking artifacts in highly compressed images. The blocking artifacts result because DWT was not applied to the entire image. For wavelet adaptations, researchers have developed spatial orientation tree adaptation
techniques that allow wavelet compressed images to be adapted to match a target data size [47], [48].

Wavelet video can also employ 3D wavelet and motion estimation techniques for inter-frame compression to reduce temporal redundancies across multiple frames. The additional computational cost and memory requirements of these techniques can be high for mobile devices. More information on 3D wavelet and motion compensation can be found in [49], [50].

As can be inferred from this section, the selection of a compression technique for streaming video from mobile phones over cell phone networks must address numerous issues. These include the computational cost, degree of adaptability, video quality, resulting video frame size, bandwidth requirements, and the impact of lost frames.

3.5 Putting it all together: Strategy to Solve the Problem

In section 3.1 the definition for real-time was selected as that of signal processing. That is, to stream video in real-time, each video frame produced by the camera should be compressed and streamed prior to the production of the next video frame. With a camera capturing video frames at 15 fps, this introduces a time limit of 0.067 s to generate, compress and stream the frame. In section 3.3, it was noted that a raw video frame can be as large as 7 Mb (920 KB) – see Table 3.1, 640x480 resolution RGB 24/YUV 4:4:4 format. As discussed in section 3.2, it was noted that cell phone network bandwidth can be widely variable. At 3 KB every 0.067 s, most video frames tend to make it through a 3G cell phone network with a minimal drop rate (about 95% of all frames can make it through). This of course will depend on a number of external factors
however such as proximity from the base station, the amount of network traffic, which itself is partially based on the time of day and day of week, and environmental conditions. Thus, the mobile device must perform some form of video compression to reduce a file as large as 920 KB to something more like 3 KB (depending on the quality of bandwidth) during this 0.067 s window.

Mobile devices are much more limited in computing power than desktop or laptop computers [51]. Because of this, it is a mistake to think that video compression techniques used to stream real-time video from computers over the Internet can be simply ported to mobile devices. The mobile devices have processors that are not only weaker than their desktop/laptop counterparts, but the amount of available processing power varies depending on background applications and battery management strategies. The battery management strategies, run by the mobile device operating systems, reduce processing speed as the battery begins to the drain. Turning off the battery management options can lead to better processor performance if one is aware enough to do this, however it does drain the mobile device battery more rapidly.

Mobile device memories are much smaller than those of desktop and laptop computers, leading to another obstacle for any mobile video compression algorithm. Although memory is typically expandable by using flash memory, this will slow down any process. Therefore, the larger flash memories may not be of much use during real-time video compression.

Another factor in streaming real-time video is that the data size of a compressed video frame itself can vary widely according to the picture content of the particular video being streamed. For instance, a video frame with a solid background can be more compressed than a video frame
with a lot of texture. Thus, the bandwidth needed to stream the compressed video frame will vary significantly from frame to frame if the video has fast changing scenes.

Fig. 3.5 shows the entire process by which the raw video frames from the camera of the mobile device undergo several stages including data reduction via RGB → YUV conversion, compression and adaptations before being streamed from the device through the cell phone data channel (3G, EDGE etc.) across the Internet to the receiving video decoder. The video frames are then decoded and displayed. The conversion from RGB to YUV and YUV to RGB can be optional, depending on the mobile device and the video application. Each video frame works its way through the process from being generated by the camera to being streamed into the cell

![Diagram of video streaming process](image)

**Fig. 3.5.** Real-time streaming of video originating from mobile devices. The raw video frames from the camera of the mobile device are compressed and adapted before being streamed via the cell phone data channel across the Internet to the video decoder where the video frames are decoded and displayed.
phone network stream before the next video frame is generated. If the receiving device is another mobile device then the following factors should also be taken into consideration: screen size, memory, processor speed, and disk space if the video is to be saved.

And so the constraints of the problem become clear. The raw video must be compressed quickly but the reduction in size must be enough to permit the video frame to be streamed without a large chance of it being dropped by the cell phone network. Even a 176x144 resolution video frame in YUV 4:2:0 format will be 38 KB in size and therefore must be compressed significantly to be streamed with confidence of not being dropped. How can these factors be reconciled?

Any application that attempts to stream video in real-time from mobile phones across cell phone networks should be able to do each of the following.

1. Compress each video frame to a size that is manageable by the cell phone network.
   a. First, convert the color space to an appropriately reduced format (e.g. from RGB 24 to YUV 4:2:0)
   b. Apply a compression algorithm that requires minimal computation, preferably using integer-only operations to compress video frames substantially in size.

2. Adapt the video transmission rate to match the available bandwidth at the time of the video streaming. This will include sensing the bandwidth or conditions of the cell phone network.

3. Determine, based on the video being streamed, how much the video should be compressed to reach a desired target size. That is, as the video itself changes, the compression level may also require changing.
The strategy described above goes beyond the capabilities of a video compression algorithm. The entire streaming process, which includes video compression, should also adapt the video transmission rate and/or control the video compression levels. These constraints and challenges will be addressed in greater detail in the following chapters.
Chapter 4  Addressing Video Compression Constraints

As described in section 1.3, before real-time video can be streamed from cell phones, the first problem to be addressed is the development of a video compression algorithm suitable for the low and variable computing resources available on mobile devices. This chapter presents the components of the mobile video compression algorithm developed in this dissertation so as to allow the streaming of live video from cell phones. This algorithm meets all the requirements of the video compression solution outlined in section 1.3.

Because of the limited resources available on mobile phones and the demands of real-time requirements, a computationally simple and efficient video compression algorithm is a critical component for the streaming of real-time video from cell phones. This part of the dissertation shows how the careful selection of video compression components can be used to strike a delicate balance between computational complexity and utilizing the limited resources available on cell phones. Here a 5-3 wavelet transformation and a new subband aligned integer run-length encoding (SAIRLE) are used for compressing wavelet coefficients. The result is a new and efficient encoding technique suitable for compressing live video streams originating from cell phones.
4.1 Introduction

In recent years, many researchers have come to agree that wavelet image compression produces better image quality than the Discrete Cosine Transform (DCT) image compression technique used in JPEG [52] and other popular video compression algorithms. For mobile devices, wavelet filters with integer coefficients are best because they do not require floating point operations and produce a suitable compression in comparison with floating point wavelet filters [10].

After wavelet transformation and quantization, some form of data coding is necessary to produce the final compressed data. Several coding schemes are available including Huffman, Arithmetic, Golomb, LZW and run-length coding. In general, run-length coding of wavelet coefficients is a natural fit for real-time mobile video encoding because it is simple to implement, requiring no probability or lookup tables, and involves less computing resources than most other data coding methods.

Alternatively, in [53] the authors present an auto-adaptive bit-plane encoding technique. This approach encodes bit level wavelet coefficients by decomposing each wavelet sub-band into quad trees. The quad tree decomposition requires multiple scanning of the same block to determine if further decomposition is necessary. This approach also requires additional memory for storing the quad trees and computation time will vary from one video frame to the next especially if the bit patterns are not sparse. In a real-time mobile environment, multiple scans of a video frame subband and additional memory are scarce resources, and this approach will most likely not be suitable for memory and time constrained real-time mobile video compression. On the other hand, the computational demands of run-length coding do not vary from frame to frame. Other researchers, [12] and [54] have studied issues relating to efficient energy
management for the compression and transmission of multimedia on mobile devices and wireless networks. Their results show that less computation translates directly into less energy usage. This dissertation utilizes run-length coding because of its simplicity and low computational overhead.

In [55] the authors showed that a subband aligned coding approach could yield significant gains in a run-length algorithm which used a fixed-length code to encode the distances between significant wavelet coefficients. However, their scheme is purely theoretical and does not yield a decodable stream. This work builds upon these broad ideas but takes a very different approach by encoding wavelet coefficients using run-length coding while aligning the scanning pattern in parallel with the directional edge detection properties of wavelet subband images. This exploits the fact that significant wavelet coefficients are aligned with the directional properties of wavelet sub-bands. In so doing, a new efficient integer run-length coding algorithm for real-time mobile video compression is described. The subband aligned integer run-length encoding (SAIRLE) algorithm produces better compressed video streams than complimentary bit-plane run-length coding schemes.

4.2 Integer 5–3 Wavelet Transformation

The 5-3 wavelet was selected for a variety of reasons. Wavelets have gained considerable attention in video compression because of their ability to produce highly compressed quality video even at very low bit rates. For real-time mobile video streaming from cell phones, it is very important to select a wavelet filter that will not be a computational bottleneck in the video compression process. The 5-3 wavelet filter was chosen because it is rational and short [56].
The 5-3 wavelet transform can be implemented with shifts and additions while avoiding multiplications [57]. Because cell phones have limited computing power with no floating point arithmetic unit, the avoidance of multiplications and the use of rational wavelet filters can significantly improve the computational speed of the wavelet transformation. Floating point operations reduce the performance of the processors of mobile devices [51]. Experiments for this dissertation utilizing commercially available cell phones and US-based mobile networks revealed that the use of rational numbers and elimination of multiplications in the wavelet transformation improved the real-time streaming video throughput by 22%. With this improvement the video encoder was able to compress each video frame efficiently and quickly enough to attain the camera’s maximum achievable frame rate of 15 fps. As such, the video encoder has been fine-tuned and scaled down to operate within the constraints of the cell phone device, achieving the necessary level of compression without overburdening the cell phone’s computational capabilities or creating a bottleneck by preventing the device from attaining its maximum 15 fps rate.

Although the lossy video compression quality obtained from longer wavelet filters such as the 9-7 is superior to the 5-3, the performance and computational gains from the short 5-3 wavelet filter in a computationally constrained cell phone outweighs such benefits. The high pass and low pass analysis and synthesis filters for the 5-3 wavelet transform is defined as follows [44]:

High pass analysis filter

\[
\tilde{d}[n] = \tilde{x}[2n + 1] + \left[ \frac{1}{2} - \frac{1}{2} \tilde{x}[2n] - \frac{1}{2} \tilde{x}[2n + 2] \right]
\]
Low pass analysis filter

\[
\tilde{s}[n] = \tilde{x}[2n] + \frac{1}{2} \tilde{d}[n-1] + \frac{1}{4} \tilde{d}[n]
\]

High pass synthesis filter

\[
\tilde{x}[2n] = \tilde{s}[n] - \frac{1}{2} \tilde{d}[n-1] + \frac{1}{4} \tilde{d}[n]
\]

Low pass synthesis filter

\[
\tilde{x}[2n + 1] = \tilde{d}[n] - \frac{1}{2} \tilde{x}[2n] - \frac{1}{2} \tilde{x}[2n + 2]
\]

where \(\tilde{x}[n]\), \(\tilde{d}[n]\) and \(\tilde{s}[n]\) are the symmetrically extended version of the input signal, high pass band output, and low pass band output respectively.

To transform a video frame, the low and high pass analysis filter is applied to the rows, and the output is stored in the left and right halves of the video frame. Then the low and high pass filters are applied again to the columns and the output is stored in the top and bottom halves of the video frame. This process yields the four subbands that are typically labeled LL, HL, LH and HH, according to the filters used for each direction of the transformation. Multi-level wavelet transformation is achieved by repeatedly applying the transformation to the lowest frequency band. After wavelet transformation, the HL, LH, and HH subbands contain wavelet coefficients with a mean of zero. Fig. 4.1 shows the multi-resolution structure of a three level wavelet transformed video frame.
4.3 Run–Length Encoding

As stated in the introduction to this chapter, run-length encoding is simple and well suited to the limited computing resources available on mobile devices. Run-length encoding can be performed on the bit-plane where wavelet coefficients are decomposed into bit-planes or on the integer values of the wavelet coefficients. Bit-plane run-length encoding is implemented by counting the runs of the 0 bits and outputting a 0 bit and the count of the runs of zeros in a 16-bit integer. All bits containing the value 1 are coded as is. In this project, integer run-length coding is performed by counting the numbers of consecutive integer zeros and outputting a 0 and then a 16-bit integer value of the runs of integer zeros. All other non-zero integer wavelet coefficients
are coded as is. In these experiments, four different scanning patterns were applied to evaluate the use of run-length coding of wavelet coefficients for integer and bit-planes.

### 4.3.1 Full Horizontal Scan

For a full horizontal scan, run-length encoding is performed by scanning each row of the wavelet transformed video frame from left to right, and from top to bottom. See Fig. 4.2.

### 4.3.2 LL First Horizontal Scan

The lowest level LL subband is first scanned and encoded in bits for the bit-plane run-length, and in 8-bit integers for the integer run-length. Then for each level, the HL, LH and HH subbands is scanned row-by-row going from left to right and top to bottom of each video frame. See Fig. 4.3.

### 4.3.3 Subband Aligned Scan

The subband aligned scanning scheme performs run-length coding by aligning the scanning path with the directional edge detection properties of wavelet subbands. The lowest level LL subband was first scanned and encoded in bits for the bit-plane run-length, and in 8-bit integers for the integer run-length. The HL subband which is sensitive to vertical edges is scanned column-by-column from left to right. Horizontal edges are highlighted in the LH subband. The LH subband is scanned in a row-by-row fashion going from top to bottom. The HH subband is sensitive to diagonal edges, but because diagonal edges are rear in real-world scenes, the HH subband
typically contains images that look more like random noise. As a result, the HH subband can be scanned horizontally or vertically with little or no impact on video compression. The HH subband is scanned in a horizontal fashion going row-by-row from top to bottom. See Fig. 4.4.

4.3.4 Zig–zag Diagonal Scan

Here the scan is made diagonally in a zig-zag pattern starting from the top left corner to the bottom right corner of the video frame just like in the JPEG DCT coefficient ordering for run-length coding. See Fig. 4.5.

![Fig. 4.2. Full horizontal scan path. Run-length encoding is performed by scanning each row of the video frame horizontally from left to right.](image-url)
Fig. 4.3. LL first horizontal scan path. The LL subband is scanned first, then for each transform level, the HL, LH and HH subbands are scanned horizontally from left to right.

Fig. 4.4. Subband aligned scan path. The LL subband is scanned first, then for each transform level the HL subband is scanned vertically while the LH and HH subbands are scanned horizontally from left to right.
4.4 Experimental Setup

Using the above scanning schemes this work examined and compared the wavelet video compression capabilities of run-length coding of the integer pixels and the bit-plane approaches in the context of real-time video on cell phones. This experiment implemented real-time video streaming software on commercially available cell phones using 5-3 wavelet transformation as described above. After transformation and quantization this experiment compared bit-plane and integer pixel run-length coding using the above scanning patterns to select the strategy that resulted in the most compressed video.

The test cell phone had a 400 MHz processor speed and 64 MB RAM. However, on fresh reboot, only 24.17 MB RAM was free due to the necessity of sharing resources with essential background services such as the phone application and other software including calendar, alarm clock, etc. Implementing the software on a real cell phone revealed these additional constraints.

Fig. 4.5. Zig-zag diagonal scan path. Run-length encoding is performed by scanning the video frame in a zig-zag diagonal pattern from top left corner to bottom right corner.
and demanded a disciplined approach to real-time mobile video streaming. Experiments were conducted using a live video of children playing in a park. The video length is 400 frames. Each video frame is of the size 240 by 320 at 8 bits per pixel.

Fig. 4.6 shows the experimental setup. Each video frame was captured live, transformed, quantized, and fed to each of the run-length coding functions. Each of the coding functions encoded the same transformed video frames. Specifically, for each video frame, a four-level wavelet transformation was utilized. In one group, a uniform scalar quantizer with step size $q=21$ was applied. In another group, the same transformed video frame was fed to a uniform scalar quantizer with step size $q=30$.

Only the six most significant bit-planes for the bit-plane coding schemes were encoded. The two least significant bit-planes were left out because they do not contain much structure and also the human eye is not able to detect much difference between 8-bit and 6-bit images [46], [28]. Note that except for the omission of the two least significant bit-planes, all eight video streams in the group with $q=21$ will produce the same higher quality video, while in the group with $q=30$ all 8 video streams will produce the same lower quality video at the decoder. Hence it is possible to evaluate the compressed data size of each run-length coding scheme by examining their average values. In these experiments, note that the 16 streams of video were neither saved nor transmitted; rather, only the data size of each video frame was computed and averaged during runtime. Also, due to memory limitations, the same memory space was reused for each coding function. More or less compression can be achieved by varying the value of $q$. As $q$ increases, the video quality degrades gracefully with blurring as a side-effect. Dynamically determining a value of $q$ for a target frame data size is beyond the scope of this particular experiment, and will be discussed in chapter 6.
Fig. 4.6. Experimental setup for comparing integer and bit-plane run-length coding of video frames. Each raw video frame was transformed with 5-3 wavelet transformation then two separate copies of the same transformed video frame was quantized to evaluate each group of run-length coding functions.
4.5 Experimental Results

Tables 4.1 and 4.2 compare the average compressed video frame data size for each run-length coding scheme for \( q=21 \) and \( q=30 \) respectively. From both Tables, row 3, column 2, contain the smallest average data size of compressed video frames encoded with the SAIRLE scheme. Overall, these experiments show that for encoding 5-3 wavelet transformed video frames, the integer run-length coding is superior to their complimentary bit-plane run-length coding schemes. The results show that for both lower and higher quality video, the subband aligned integer run-length encoding (SAIRLE) of wavelet coefficients produced the smallest size of video stream.

Table 4.1.
Average data size of compressed 400 240x320 pixel live video frames with \( q=21 \)

<table>
<thead>
<tr>
<th></th>
<th>Bit-Plane (bytes)</th>
<th>Integer (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Horizontal</td>
<td>9065</td>
<td>4716</td>
</tr>
<tr>
<td>LL First Horizontal</td>
<td>8684</td>
<td>4710</td>
</tr>
<tr>
<td>Subband Aligned</td>
<td>6347</td>
<td>3998</td>
</tr>
<tr>
<td>Zig-zag Diagonal</td>
<td>13025</td>
<td>6457</td>
</tr>
</tbody>
</table>

Table 4.2.
Average data size of compressed 400 240x320 pixel live video frames with \( q=30 \)

<table>
<thead>
<tr>
<th></th>
<th>Bit-Plane (bytes)</th>
<th>Integer (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Horizontal</td>
<td>5764</td>
<td>2917</td>
</tr>
<tr>
<td>LL First Horizontal</td>
<td>5363</td>
<td>2908</td>
</tr>
<tr>
<td>Subband Aligned</td>
<td>3549</td>
<td>2327</td>
</tr>
<tr>
<td>Zig-zag Diagonal</td>
<td>8709</td>
<td>4357</td>
</tr>
</tbody>
</table>
4.6 More on SAIRLE

The SAIRLE scheme is significant in that it performs run-length encoding by aligning the scanning path with the edge detection properties of wavelet sub-bands so that more compression can be achieved. SAIRLE compresses video by taking advantage of the zero mean property of transformed wavelet coefficients and by aligning the scan path with the edges, more consecutive zeros can be scanned and encoded. Hence more compression is achieved.

To provide a visual representation, Fig. 4.7 shows a sample video frame, and Fig. 4.8 displays a two-level 5-3 wavelet transformed version of the sample video frame. For each level of wavelet transform, notice in Fig. 4.8 that the LH and HL sub-bands of the transformed video frame highlight the horizontal and vertical edges of the original picture in the video frame. The HL, LH, and HH sub-bands also contain wavelet coefficients with a mean of zero. The LL sub-band contains a miniaturized reproduction of the original picture in the video frame.

Fig. 4.7. Sample video frame. This video frame contains a picture with a background featuring vertical and horizontal edges.
**Fig. 4.8.** A two-level 5-3 wavelet transformed copy of the sample video frame from Fig. 4.7. For each transform level, the HL subband highlights vertical edges, horizontal edges are pronounced in the LH subband and the HH subband is sensitive to diagonal edges.

### 4.7 Summary

This chapter has shown that the careful selection of video compression components to minimize computational complexity can significantly impact the performance of real-time video streaming on resource constrained cell phones. 5-3 wavelet transform and subband aligned integer run-length encoding (SAIRLE) algorithm are able to efficiently compress real-time video originating from cell phones to attain a maximum achievable 15 fps video stream.

In this chapter the components of a scaled down video compression algorithm suitable for real-time video compression for maximum performance on mobile devices have been presented. Now that a real-time video compression algorithm has been developed which allows the video to be efficiently compressed to a manageable data size, the next set of challenges to be addressed is the constraints of streaming the compressed video over the low and variable bandwidth of cell phone networks.
Every problem has in it the seeds of its own solution. If you don't have any problems, you don't get any seeds.

- Norman Vincent Peale

Chapter 5  MoStVid: Developing an Adaptive Framework for Streaming Real–Time Video from Mobile Devices

5.1 Introduction to MoStVid

The 5-3 Wavelet Transform and SAIRLE video compression algorithm presented in chapter 4 clearly demonstrate that streaming video transmitted from a cell phone across a cell phone network is possible. However, due to the variability and limitations of cell phone network bandwidth, the streaming of video can still result in many dropped frames, thus impacting the quality of the received video. There are a number of mechanisms that can positively impact the quality of the real-time video reception. For instance, reducing the number of transmitted frames per second (fps) could provide a smoother video. Alternatively, reducing the data size of the video frames may improve the transmission rate while not necessarily damaging the video quality as perceived by the recipient. In response to these challenges, this chapter presents a real-time decision making algorithm for Mobile Streaming Video (MoStVid) which adapts the fps rate of the video camera and utilizes the wavelet video compression algorithm described in chapter 4.

---

1 © 2009 IEEE. Reprinted, with permission, from Proceedings of 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Improving Delivery Rate for Real-Time Mobile Streaming Video, N. V. Uti and R. Fox.
In this chapter, a version of MoStVid is presented which adapts the fps rate based on the current bandwidth of the cell phone network, and the data size of the video frames. Using artificial intelligence, the algorithm makes real-time decisions to improve the video delivery rate. Specifically, feedback is provided from the video recipient in terms of the number of video frames received per second. From the transmitting cell phone, the average data size of each video frame is determined. A simple yet effective decision making algorithm using artificial intelligence techniques then determines whether the number of frames transmitted per second should be increased, decreased or held steady.

An additional decision making option is to increase or decrease the video frame data size by applying various lossy compression techniques. This can provide MoStVid with a greater flexibility in deciding how best to transmit the video. This chapter describes preliminary work on the MoStVid decision making algorithm, which here is limited to adjusting frames transmitted per second, and presents some experiments and results. The summary of this chapter will address the idea of adjusting video frame data size.

### 5.2 Real–Time Decision Making

While the 5-3 Wavelet Transform and SAIRLE video compression technique described in chapter 4 permit real-time streaming video from a cell phone over a cell phone network, they do not guarantee that all video frames will be received. In fact, in experiments run at varying times of the day and week, with frame data sizes ranging from 380 bytes to 1,200 bytes, the video frame delivery rate can be poorer than 50%. That is, fewer than half of the transmitted frames are actually received.
What can be done to improve performance? Obviously the cell phone network is impacted by any number of external circumstances. The cell phone’s video compression algorithm can only control what is being transmitted. This is limited to controlling:

1. the number of video frames per second transmitted,
2. the data size of the video frames being transmitted.

A dynamic decision making algorithm can be used to control these two aspects. Periodically, the recipient (which is running the video decoding algorithm) sends back across the network the number of video frames received. If the recipient is receiving fewer frames than is desired, the transmitting cell phone may decrease the number of frames transmitted per second in an attempt to reduce the needed bandwidth while still retaining a certain degree of quality in the received video. For this chapter, a real-time decision making algorithm for MoStVid has been implemented which only seeks to increase or decrease the number of video frames transmitted per second. This algorithm uses simple yet suitable artificial intelligence decision making techniques based on the current state of the cell phone network and the data size of recent frames transmitted. In the next chapter, this project will look at varying the data size of each video frame by increasing or decreasing the amount of loss during compression.

In order to decide how best to continue transmitting video frames, MoStVid examines two pieces of information. At ten second intervals, MoStVid then decides how to adjust the video transmission rate in an attempt to improve the number of frames received per second. The first piece of information is the number of frames received versus the number of frames transmitted over several ten-second time intervals. This provides a view of the current state of the cell phone network, that is, of the available bandwidth. Fig. 5.1 demonstrates several ten-second intervals
showing that the initial 15 frames per second transmission rate was far too high for the current state of the cell phone network, resulting in a need to lower the number of frames transmitted per second. The second piece of information used is the data size of the video frames being transmitted, on average over several ten-second time intervals. The data size of each video frame fluctuates depending on the content of the frame. It is highly unlikely that any two frames will have the same data size. Fig. 5.2 illustrates how variable the frame data size is.

Because of fluctuating bandwidth and fluctuating video frame data sizes, any particular bandwidth estimation or video frame data size may be atypical. As a result, at ten-second intervals, the following exponential weighted moving average (EWMA) formulas (adapted from [58]) are used to estimate typical frame drop rate, typical frame data size, and their corresponding variation.

**Fig. 5.1.** Transmitted fps versus delivered fps. The initial large disparity between the transmitted fps and the delivered fps is reduced as MoStVid adjusts the fps rate of the transmitted video to be close to the bandwidth capacity of the cell phone network.
Each EWMA is a weighted combination of its previous value and the current value. The recommended values for $\alpha$ is 0.125 and $\beta$ is 0.25 [58]. The variables, $\alpha$ and $\beta$, allow more
current values to have a higher influence on the estimated values. This results in a smoother estimate of the most recent state of the cell phone network and video frame data size while also accounting for the history of previous values. The $DropVarThreshold$ (from equation 2) is a threshold used in the decision making to determine if the percentage of dropped frames is increasing, steady or decreasing. The $DataSizeVarThreshold$ (from equation 4) is also used in the decision making to determine if the data size of recent frames is increasing, steady or decreasing.

If the percentage of frames being dropped is significant, (outside of the estimated $DropVarThreshold$) and the data size of the frames is relatively large (outside of the estimated $DataSizeVarThreshold$), then the number of frames being transmitted per second will be reduced. As time goes on, if frames continue to be dropped, the number of frames being transmitted will continue to be reduced. If the average data size of the frames being transmitted has been increasing over recent time intervals, then the number of frames being transmitted will be reduced (for the time being). If the number of frames being received increases (perhaps because the cell phone network bandwidth is improving), then the number of frames transmitted per second is increased. If the average data size of the frames being transmitted has been decreasing and the number of frames being received remains static or increases, then the number of frames transmitted is increased. The maximum number of frames transmitted per second is limited to 15.

Table 5.1 illustrates the decision making criteria and how the conditions are translated into actions (by increasing, maintaining, or decreasing the number of frames per second). Notice that if the number of frames being dropped is increasing, the number of frames transmitted decreases even if the frame data size is decreasing. The rationale here is to ensure that as many video
frames are received as possible in spite of the data size. On the other hand, if the number of frames being received is steady and the video frame data size is decreasing, MoStVid takes advantage of this by increasing the number of frames transmitted per second. Finally, notice that the adjustment to the number of frames transmitted per second can be in increments/decrements of 0.5 instead of only whole numbers. The actual adjustment being made is in the sampling rate of the video frames. Thus on average the number of frames transmitted might be changing by +/- 0.5 per second, not that a half of a frame is transmitted during a particular time interval.

**Table 5.1.**
Decision making criteria and actions for MoStVid

<table>
<thead>
<tr>
<th>Percentage of dropped frames</th>
<th>Data size of frames</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 10% or Increasing</td>
<td>Increasing</td>
<td>Decrease fps by 1.0</td>
</tr>
<tr>
<td>&gt; 10% or Increasing</td>
<td>Steady</td>
<td>Decrease fps by 1.0</td>
</tr>
<tr>
<td>&gt; 10% or Increasing</td>
<td>Decreasing</td>
<td>Decrease fps by 0.5</td>
</tr>
<tr>
<td>Steady</td>
<td>Increasing</td>
<td>Maintain fps</td>
</tr>
<tr>
<td>Steady</td>
<td>Steady</td>
<td>Maintain fps</td>
</tr>
<tr>
<td>Steady</td>
<td>Decreasing</td>
<td>Increase fps by 0.5</td>
</tr>
<tr>
<td>Decreasing</td>
<td>Increasing</td>
<td>Increase fps by 0.5</td>
</tr>
<tr>
<td>Decreasing</td>
<td>Steady</td>
<td>Increase fps by 1.0</td>
</tr>
<tr>
<td>Decreasing</td>
<td>Decreasing</td>
<td>Increase fps by 1.0</td>
</tr>
</tbody>
</table>
5.3 Experimental Setup and Results: Testing the Performance of MoStVid

In order to demonstrate the performance of MoStVid, a test cell phone with a 400 MHz processor speed and 64MB RAM was used. On fresh reboot, only 24.17MB RAM was actually free due to the necessity of sharing resources with essential background services such as the phone application and other software including calendar, and alarm clock.

Experiments were conducted using various live video from children playing in a park. The length of each experimental run varied but is approximately 6,000 frames and as long as 15 minutes. MoStVid adjusts the number of transmitted frames per second at ten-second intervals based on receipt of feedback from the video decoder, being asynchronous, the recipient video decoder is not necessarily responding in exact ten-second intervals. Each video frame is of the dimension 176 by 144 at 8 bits per pixel.

In the experiments reported in this chapter, the data size for the video frames ranged from 433 to 680 bytes with an average data size of 561.3 bytes. Video frames are sent using UDP over an EDGE mobile network.

Fig. 5.3 illustrates the result of an example run of MoStVid with real-time decision making. The figure shows the percentage of video frames delivered during each ten-second interval of the experiment. Initially, as few as 42% of the transmitted video frames were received. After the first ten-second interval, the decision making algorithm begins to adjust the number of video frames transmitted. As shown in this figure, the ratio of video frames received divided by the video frames transmitted improves. At first, the ratio was about 1:2, but approached 1:1 by the
time that particular experiment was completed. Over time, the adjustment brings the cell phone transmission rate into alignment with the capability of the cell phone network as shown in Fig. 5.3.

Fig. 5.4 shows the transmitted fps per ten-second interval over the same time period as Fig 5.3. Notice that the transmission begins with 15 frames per second but is quickly reduced in response to the cell phone network loss of roughly half of the transmitted video frames. The system settled at around 8-9 frames per second.

Also computed was the accumulated percentage of frames received from those transmitted. That is, the percentage of received video frames for each ten-second interval were accumulated to keep a running total of received frames. The average by the end of the experimental runs was around 80% (having started at near or under 50% in some of the experiments). In the experiment shown in Fig. 5.5, the cumulative delivery rate improved by 40% from an initial 42% to a final 82% success rate.
**Fig. 5.4.** Transmitted fps over the same time interval as Fig. 5.3. The fps rate is reduced from 15 fps to the between 8 and 9 fps which is allowed more frames to be successfully delivered.

**Fig. 5.5.** Percent of frames delivered and cumulative percent of frames delivered. As MoStVid adjusts the fps rate of the video, the cumulative effect is that more and more video frames are delivered.
Fig. 5.6 shows a graph of data collected for another live video streaming experiment which lasted for 15 minutes. The jagged curve shows the percentage of frames delivered every 10 seconds. The smoother curve shows the cumulative percentage of frames delivered. This graph demonstrates the ability of the MoStVid decision making algorithm to dynamically adjust the video frame delivery rate, from an initial frame delivery rate of 49.3% to achieve a final frame delivery success of 77.6% over the course of the experiment. This improvement highlights MoStVid’s ability to improve total frame delivery rates by adjusting the fps rate.

Fig. 5.7 shows the dynamic fps adjustment (of the same video from Fig. 5.6) in response to the video frame delivery feedback from the decoder. Starting from an initial video rate of 15 fps with a 49.3% delivery rate, the algorithm reduced the video rate down to between 6 and 10fps to eventually achieve a cumulative delivery success rate of 77.6%. The fact that a 15 fps rate was never achieved again shows that the network was not capable of delivering 15 fps.

Fig. 5.6. Percent of frames delivered and cumulative percent of frames delivered. More video frames are delivered as MoStVid adjusts the fps rate to be more aligned with the bandwidth.
In the following paragraphs, the statistical data from this example is used to analyze the potential bandwidth of the network in terms of average frame data size. Table 5.2 shows that at an average frame data size of 476 bytes, with sizes ranging from 380 to 653 bytes, the network could deliver an overall maximum of 77.6% of the video frames, with video rates ranging from 6 fps to 15 fps. This average is important because it points out the ability to determine the bandwidth of the network so as to use this information to improve MoStVid later in the project. Being able to dynamically detect the capacity of the network means that MoStVid can be improved to also control the compressed video frame data size to be more aligned with the bandwidth.

Table 5.3 shows the statistics of the subset of video frames delivered at 100% success rate. From this table it can be seen that at an average frame size of 455 bytes, with data sizes ranging
from 380 to 605 bytes, the network is able to deliver 100 percent of the frames at a video rate of 6 fps – 9 fps. The average frame delivery of 455 bytes supports the idea that if frame sizes remained within a standard deviation of 43.1 (from Table 5.3) from the overall average frame size of 476.1 (in Table 5.2), then the delivery rate can be close to 100%.

### Table 5.2.
Statistics of all frame data sizes in this experiment

<table>
<thead>
<tr>
<th>Frame Data Size (bytes)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Frame Size</td>
<td>653.5 Video rate was 8 fps, delivery rate was 62%</td>
</tr>
<tr>
<td>Min Frame Size</td>
<td>380.5 Video rate was 9 fps, delivery rate was 100%</td>
</tr>
<tr>
<td>Average Frame Size</td>
<td>476.1</td>
</tr>
<tr>
<td>Median Frame Size</td>
<td>472.8</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>52.9</td>
</tr>
</tbody>
</table>

### Table 5.3.
Data size statistics of video frames with 100% delivery success

<table>
<thead>
<tr>
<th>Frame Data Size (bytes)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Frame Data Size</td>
<td>605.9 Video rate was 7fps</td>
</tr>
<tr>
<td>Min Frame Data Size</td>
<td>380.5 Video rate was 9fps</td>
</tr>
<tr>
<td>Average Frame Data Size</td>
<td>455.1</td>
</tr>
<tr>
<td>Median Frame Data Size</td>
<td>448.6</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>43.1</td>
</tr>
</tbody>
</table>
Table 5.4 provides a video rate summary of the data sizes of video frames with 100% delivery. This reveals that for this experiment, the network can support a 9 fps video rate at an average data size of 398 bytes, 8 fps video rate at 409 bytes, 7 fps video rate at 443 bytes, and 6 fps video rate at 460 bytes. This shows that if MoStVid can dynamically balance both the fps and the average data size of the video frames, then a near 100% delivery rate is attainable.

In summary, a total of 25 experiments were conducted. In these experiments the improvement in the cumulative delivery rate ranged from 18% to 43% with an average improvement rate of 32%. The experiments show that the decision making algorithm permits MoStVid to improve its performance. It should be noted that because video frames were being lost by the network, reducing the number of frames transmitted does not result in a worse quality video stream, but rather an improved one because a greater number of video frames are received at a smoother rate.

**Table 5.4.**
Video rate summary of the data sizes of video frames with 100% delivery success

<table>
<thead>
<tr>
<th>Video Rate (fps)</th>
<th>Average Frame Data Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>398.9</td>
</tr>
<tr>
<td>8</td>
<td>409.5</td>
</tr>
<tr>
<td>7</td>
<td>443.7</td>
</tr>
<tr>
<td>6</td>
<td>460.1</td>
</tr>
</tbody>
</table>
5.4 Summary

This chapter shows that real-time streaming video from cell phones and cell phone networks is achievable. A wavelet video compression algorithm can greatly reduce the data size of video frames to something manageable for both the processing power and available memory of a cell phone and the bandwidth of a cell phone network. However, due to bandwidth variability and the error-prone nature of cell phone networks, a large percentage of video frames can be dropped between transmission and reception. This chapter has described how decision making techniques can be used to enhance the video compression algorithm. As such, a real-time decision making component has been added. The experimental results presented in this chapter show that the number of dropped frames can be significantly decreased through the intelligent and dynamic adjustment of the fps rate.

Further, analysis of the range of video frame data sizes which were successfully delivered at rates of 100% show the potential for MoStVid to attain close to 100% frame delivery rate. This can be achieved if the ability to control the frame data size to be within a dynamic target frame data size for the capacity of the network can be implemented. This observation will be explored further in chapter 6.
Painting in watercolor is like walking a tight-rope; one must achieve a perfect balance between what the paint wants to do and what the artist wants to do, or all is lost.

- Mary C. Taylor

Chapter 6 Expanding MoStVid: Adapting Video Frame Rates and Data Sizes for Enhanced Video Quality

6.1 Introduction

The previous chapter showed the potential of MoStVid to improve video delivery rates. This chapter explores additional features of the adaptive MoStVid framework and how they relate to cell phone network conditions, the video being streamed, and mobile computing resources on the mobile device. Presented here are several enhancements to the MoStVid framework, including the dynamic adjustment of the video quality and fps rate.

This chapter builds upon the findings and techniques presented in the previous two chapters to develop a more comprehensive and dynamic system which can adapt video compression levels and transmission rates to changing network conditions. The framework attains all aspects of the solution described in section 1.3, in that it addresses cell phone computational and memory constraints, real time video compression constraints, and cell phone network constraints. The updated MoStVid implements a coherent and fully featured adaptive framework for real-
time mobile streaming video. Significantly, this chapter also describes and reports on a variety of bandwidth patterns identified from extensive new experiments streaming live video on real cell phone networks. An understanding of these bandwidth patterns is important to the proper design and implementation of adaptive and dynamic frameworks for the delivery of video over cell phone networks.

The expanded MoStVid framework discussed here achieves close to 100% video delivery rate on most of the live streaming video experiments. This chapter provides a complete description of the MoStVid framework including a thorough examination of the various impacts that the cell phone network and limited computing resources of the cell phone can have on streaming live video.

6.2 Methodology on Mitigating Dropped Video Frames

By default, the Internet offers a best-effort delivery service to packets arriving at a router queue. All packets are treated with equal priority on a first-in-first-out basis, including those containing time and delay-sensitive real-time video. In the event of congestion at a router, packets are queued and processed with delay; if the router’s buffer is full, packets are dropped. The issue of router delay and packet loss also applies to video packets that are streamed via cell phone data channels to the Internet. Given the rigid time constraints on real-time streaming video, even a moderately congested router can cause delay and packet loss that can quickly lead to unacceptable video quality [9]. The situation is worse for mobile devices connected to the
Internet via limited bandwidth and error-prone wireless channels. Video frame drop rate can be in excess of 50%.

For cell phone networks with limited and variable bandwidth, it is very important to address the problem at its root by controlling excess traffic (i.e. additional packets) from getting into the network in order to minimize the waste of bandwidth that results from packet loss. Where possible, it is more beneficial to avoid sending packets into a congested network because dropped packets waste bandwidth. Most video encoders assume that if video is encoded at a certain bit rate, the network should be able to deliver the video as long as it is encoded for the rated bandwidth. However, it has been shown that due to wireless bit errors and variable latency, wireless networks can have a measured throughput that is much less than the rated bandwidth [59].

To reemphasize a critical point, the delivery of real-time video over cell phone networks must deal with three sets of interacting constraints: variations in bandwidth; variations in video frame data size; and variations in the computing resources available to the cell phone. In addition to the bandwidth constraints identified above, available wireless bandwidth can vary significantly according to the user’s physical location, mobility, and environmental conditions [21]. Video frame data size can vary from frame to frame depending on picture content. Finally, the processor speed and memory of the cell phone are also variable according to the demands of background processes, power consumption and battery management systems. Because of the complexity of this situation, non-adaptive solutions to the streaming of video over cell phone networks cannot consistently be successful. Instead, only a dynamic and continuously updated framework which takes into account all of these constraining factors can provide the intelligent system necessary to adapt constantly to changing network, video, and cell phone conditions.
Unlike stored video waiting to be broadcast, real-time video has the advantage that the video is still being captured live and encoded while earlier video frames are streamed into the network. Hence, the video frame rate and video quality can be adapted if information about the state of the network is known. In this chapter, an updated adaptive real-time Mobile Streaming Video (MoStVid) framework is presented. This enhanced version of MoStVid continuously takes into account variations in video frame data size, the computing resources available on the cell phone, and network conditions. As such, MoStVid is aware of the conditions that cause packets to be delayed or dropped in wireless and mobile networks. These conditions can take three forms [9]:

1. delay and loss due to congestion and buffer overflow in routers in the wired portion of the network,
2. bit error corruption and loss in wireless links,
3. loss and delay due to mobility handoff (transfer of an on-going call from one base station to another).

To be successful, any bandwidth-sensitive applications, especially real-time mobile streaming video applications transmitting video over wireless links, must be aware of these conditions.

The updated MoStVid presented in this chapter is an application layer framework which uses decision making and feedback from the video receiver to model network bandwidth and respond to changing network conditions. It does so by adjusting the video transmission rate in frames per second (fps) and video frame data size to best utilize available bandwidth. The result is an adaptive framework for real-time mobile streaming video that provides significant improvement on bandwidth usage, enabling a high delivery rate of quality video.

This research differs from previous works related to adaptive real-time video streaming (e.g., [11], [60]) in four important aspects. First, instead of using computers, stored video, or
simulated networks, this research cites experiments performed entirely with live streaming video on actual cell phone networks using commercially available cell phones as video sources. Second, through these experiments, MoStVid’s decision making rules were tailored to handle a variety of naturally occurring cell phone bandwidth patterns. Third, the averaged quantization values of the wavelet coefficients (to adapt video compression levels) have been reported as a measure for the video quality adaptations. The commonly used peak signal-to-noise ratio (PSNR) metric is not practical for use in real-time video experiments originating from cell phones as it requires the original uncompressed raw video frames to be saved on the device and later used for the PSNR computation.

Further, at very low video bit rates, the PSNR metric is not indicative of actual video quality [11], [61]. Finally, this work is the first academic project to report a complete solution to address the problem of streaming real-time video originating from cell phones and over cell phone networks. Implementing the software on real cell phones and real cell phone networks points out limitations in the available resources and the necessity for computational efficiency, simplicity and careful memory management. While improvements are being made in the hardware of cell phones and bandwidth, the cell phones and bandwidth that are currently considered high-end will continue to be used in most parts of the world for many years to come.

The rest of this chapter is organized as follows. In section 6.3, the adaptive wavelet video compression algorithm is described. Section 6.4 presents the full MoStVid framework including the decision making aspects, the challenges of balancing the tradeoff between video compression levels and fps rates, mobile-network-condition awareness features, and adaptations to limited mobile computing resources. Sections 6.5 and 6.6 provide experimental setup, and results. Finally, section 6.7 offers summaries.
6.3 Adaptive Wavelet Video Compression

In order to support a high video frame delivery rate of real-time video streamed over the limited and variable bandwidth of cell phone networks, the raw video frames must be adaptively compressed so as to match the variable bandwidth of cell phone networks. Adaptive Video compression is achieved through 5-3 discrete wavelet transform (DWT), quantization, and symbol encoding using sub-band aligned integer run-length encoding (SAIRLE). A high level overview of the wavelet video encoder is depicted in Fig. 6.1. From Fig. 6.1, raw video frames undergo three stages to produce compressed video frames. The 5-3 DWT and the SAIRLE stages of the video compression algorithm are described in chapter 4. This chapter expatiates on how video compression levels can be adapted in the wavelet video encoder.

Wavelet coefficients can be adapted by using Spatial Orientation Trees (SOT) [62], [47], [48] or by introducing a large quantization interval around zero – which is called dead zone quantization [46]. This research utilizes the dead zone quantization because of its simplicity.

**Fig. 6.1.** Wavelet video encoder. Raw video frames undergo three steps to produce compressed video frames: 5-3 DWT, quantization, and symbol encoding using sub-band aligned integer run-length encoding (SAIRLE).
The computational cost of the dead zone quantization technique is a constant. The dead zone quantization simply identifies the wavelet coefficients within the quantization threshold $q$ and zeros them out. This quantization approach adapts the video as desired and is suited to the limited computational resources of cell phones.

### 6.3.1 Quantization and Adaptive Video Qualities

After wavelet transformation, the resulting wavelet coefficients are quantized. For the dead zone quantization which is employed here, the wavelet coefficients with intensities $\pm q$ around zero are truncated to zero in order to achieve more compression. The zero dead zone quantization is not reversible. Zeroed wavelet coefficients result in lossy compression because distortions are introduced at the video decoder. Let $V$ be the transformed video frame containing the wavelet coefficients $w_{ij}$. Then the subset of non-recoverable quantized detail wavelet coefficients, $D(.)$ is given by

$$D(q) = \{(i,j) \in V \mid |w_{ij}| < q\}. \tag{1}$$

Quantization $Q(.)$, is achieved using

$$Q(w,q) = \begin{cases} w_{ij}, & \text{for } |w_{ij}| \geq q \\ 0, & \text{for } |w_{ij}| < q. \end{cases} \tag{2}$$
From (2), $q$ dictates the level of lossy compression from frame to frame. Smaller values of $q$ result in less compression, higher video frame quality, and video frames of larger data sizes. Larger values of $q$ result in more compression, lower video frame quality, and video frames of smaller data sizes. The value $q$ can be dynamically adjusted for each video frame so that the compressed video frame data size meets a desired target size.

When provided with a target frame data size, the wavelet video encoder uses a heuristic to generate a $q$ value that results in a compressed video frame data size to match the provided target frame data size. The encoder looks at the $q$ values of recently compressed video frames and their corresponding data sizes, then either increases or decreases $q$ before compressing a new video frame. Given the time-sensitive constraints on real-time video and the limited computing resources on the mobile device, the video encoder cannot afford to invest complex computation to analyze each video frame before the generation of $q$ to match a target frame data size. This simple heuristic is very effective in predicting the $q$ value that can result in the desired target frame data size because consecutive video frames typically contain similar pictures.

The heuristic is also limited in that the resulting data size of a compressed video frame is determined by the particular picture content of the frame. After compression, video frames with complex pictures can result in larger data sizes that may not match the target frame data size for the bandwidth. Consequently, compressed video frames containing less textured pictures (such as blank walls) can result in data sizes that are below the target frame data size. Hence, the data sizes of compressed video frames are dynamic.

Because the dead zone quantization simply converts to zero a range of intensities of wavelet coefficients that are $\pm q$ around zero, it is possible to use $q$ for a quantitative evaluation of the
resulting video quality. $q$ can range from 12 to 35. The 400 frame Foreman video sequence [63] and the 300 frame Stefan video sequence [64] are well-known YUV video sequences that are commonly used by researchers. Fig. 6.2 and Fig. 6.3 show snapshots of frame #199 of the Foreman video sequence and frame #1 of the Stefan video sequence. In these figures, the original video frames are shown along with the same video frames quantized at various $q$ values. From these sample frames, it can be seen that lower $q$ values result in better quality video frames because less intensities are lost. High $q$ values result in poorer quality video frames because more intensities are lost.

![Fig. 6.2. Frame #199 of Foreman video quantized at various $q$ values.](image)

(a) Original (b) $q=12$ (c) $q=18$ (d) $q=22$ (e) $q=26$ (f) $q=31$ (g) $q=33$ (h) $q=35$. The quality of the video frames degrades with increasing values of $q$. At smaller $q$ values, the eyes of the Foreman are clearly visible and well defined, but blur and become less distinct as $q$ increases.
Fig. 6.3. Frame #1 of Stephan video quantized at various $q$ values. (a) Original (b) $q=12$ (c) $q=18$ (d) $q=22$ (e) $q=26$ (f) $q=31$ (g) $q=33$ (h) $q=35$. Notice that as $q$ increases the advertisements in the background become less legible and Stephan’s racket becomes a smudge.

Table 6.1 summarizes how the value of $q$ can impact individual video frame quality and the resulting video quality. When $q$ is near 12 the video frames are close to uncompressed video frame quality. Experiments show that it is not beneficial to select $q$ to be less than 12 as the data sizes of the video frames will be larger while the video frame quality will not appear to be much improved. This is because the value of 12 for $q$ has been generalized from the near-optimal wavelet coefficient shrinkage formula in the famous paper by Donoho et. al [65].

At $q$ values nearing 35 the video frame is of low quality, but the resulting video should still be acceptable. This lower limit was put in place to ensure that the encoder does not produce video frames of unacceptable quality. In this context, low quality video frames, as described in Table 6.1, still produce acceptable quality video when streamed at high frame rates and delivered
at high percentages. Due to this lower limit the video encoder is guaranteed to produce useful video as long as the video frames are delivered successfully.

An advantage of using the dead zone quantization is that it provides a very reliable and computation-free way of detecting video quality in real-time. In real-world cell phone bandwidth, it is not always desirable for the encoder to match the target frame data size, as different videos produce different quality video frames at the same target data size. When the 5-3 DWT is used for any video frame, if $q$ is approaching 35, the quality of video frames will be low regardless of the resulting compressed data size. As such, the dead zone quantization provides a highly reliable real-time performance matrix to allow the encoder to detect video quality and not stream unacceptable video for any target frame data size.

### Table 6.1.
Summary of How Adjustments in $q$ Affect the Resulting Video Quality

<table>
<thead>
<tr>
<th>$q$</th>
<th>Video Frame Quality</th>
<th>Resulting Video Quality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 – 17</td>
<td>Excellent</td>
<td>High to very high</td>
<td>Excellent video frame quality produces close to uncompressed video quality.</td>
</tr>
<tr>
<td>18 – 21</td>
<td>High</td>
<td>Very good to high</td>
<td>High video frame quality produces very good quality and sharp video.</td>
</tr>
<tr>
<td>22 – 25</td>
<td>Good</td>
<td>Good to very good</td>
<td>Good video frame quality can produce very good video.</td>
</tr>
<tr>
<td>26 – 30</td>
<td>Passable</td>
<td>Acceptable to good</td>
<td>Passable video frame quality still produces more than acceptable quality video.</td>
</tr>
<tr>
<td>31 – 35</td>
<td>Low</td>
<td>Acceptable</td>
<td>Lowest acceptable video frame quality still produces useful video.</td>
</tr>
</tbody>
</table>
6.4 MoStVid

Mobile Streaming Video (MoStVid) is a framework to provide an adaptive, robust, and scalable streaming video environment for mobile devices. The main goal is to deliver a high percentage of video frames at as high a frame rate as possible so as to provide useful video despite constrained computing resources and limited and varying bandwidth.

There are three factors that contribute to the perceived quality of real-time streamed video: the individual frame quality, the frame rate in frames per second (fps), and the percentage of frames delivered. The individual video frame quality is controlled in the wavelet video encoder. The fps rate is optimized to fit the network bandwidth as sensed by using feedback from the recipient of the streaming video. Through these changes, MoStVid is able to influence the percentage of video frames delivered such that it improves as much as possible. Further, the wavelet video encoder is adaptive and can gracefully adjust video compression levels. MoStVid uses this adjustment by providing the video encoder with a target data size for each frame. The video encoder can adjust the video compression level accordingly to match the target frame data size. The following sections describe the MoStVid framework, decision making engine, the tradeoff between fps and video compression level, mobile-network-condition awareness features, and features to support the limited mobile computing resources.

6.4.1 The MoStVid Framework

As shown in Fig. 6.4, MoStVid is an application layer solution to the problem of real-time streaming video from mobile devices over cell phone networks. The mobile device’s video
camera produces raw video frames based on a sampling rate as determined by MoStVid (this is explained below). The wavelet video compression scheme from chapter 4 is used to transform the raw video into lossy compressed video frames for transmission. The compressed video frames are encapsulated in *Real Time Protocol (RTP)* packets and sent with UDP through the cell phone data channel and across the Internet to the video receiver.

The video receiver decodes the compressed video frames using a reverse process from that explained in section 6.3. The decompressed video frames are then displayed on the receiving device (whether that is another mobile device, computer, server or other). In addition, the decoder keeps track of the number of frames delivered per second, and in five-second intervals sends this information back to the transmitting mobile device for feedback.
Fig. 6.4. The MoStVid Framework. The MoStVid decision engine controls the target frame data size of video frames and the fps sampling rate at which video frames are transmitted to the cell phone network.
6.4.2 Decision Making Engine

The field of Artificial Intelligence (AI) provides a variety of well known problem solving techniques to aid in decision making [66]. AI rules can be developed to support decision making for solving problems that require a decision maker to select actions in real-time to provide dynamic response and deal with uncertainties in decision problems [67].

MoStVid uses three sets of data for dynamic decision making to deal with the uncertain bandwidth behavior of cell phone networks. The first piece of data is the estimated bandwidth. MoStVid tracks the trend of bandwidth using the feedback of the number of video frames delivered. The number of video frames delivered is subtracted from the number of frames sent in the same interval to compute the current frame drop rate. The trend in bandwidth can be steady, increasing, or decreasing. MoStVid uses the following exponentially weighted moving average (EWMA) formulas to compute a smoother estimate of typical frame drop rate and its variation. These formulas were adapted from the EWMA formulas used by TCP for computing retransmission time [68].

\[
\text{EstDropPercent} = (1 - \alpha) \times \text{EstDropPercent} + \alpha \times \text{CurrentDropPercent}
\]

(3)

\[
\text{DropVarThreshold} = (1 - \beta) \times \text{DropVarThreshold} + \beta \times |\text{CurrentDropPercent} - \text{EstDropPercent}|
\]

(4)

where \(\alpha\) is 0.125 and \(\beta\) is 0.25. The \text{EstDropPercent} and \text{DropVarThreshold} from (3) and (4) maintain a weighted moving average of the frame drop rate and variation. The parameters \(\alpha\) and \(\beta\) are weighting factors which allow more recent values to have a greater influence than older values.
The second piece of data used in decision making is an EWMA of the video frame data sizes. MoStVid tracks the trend in the data sizes of video frames using a moving average of the data sizes of compressed video frames. Compressed video frame data sizes vary depending on the picture content of the video frame. Because consecutive video frames typically contain similar pictures, the trend in the data sizes of compressed video frames can be relatively steady, increasing, or decreasing. As with (3) and (4), a smoother estimate of average video frame data size and its variation is computed as:

\[ \text{EstAvgFrameDataSize} = (1 - \alpha) \times \text{EstAvgFrameDataSize} + \alpha \times \text{CurrentAvgFrameDataSize} \]  

\[ \text{DropVarThreshold} = (1 - \beta) \times \text{DropVarThreshold} + \beta \times |\text{CurrentAvgFrameDataSize} - \text{EstAvgFrameDataSize}|. \]  

The above EWMA formulas of the data sizes of compressed video frames effectively supply MoStVid with information on the trend of the data sizes of video frames. The \text{DropVarThreshold} of (4) is used in the decision making to determine if the bandwidth is steady, increasing, or decreasing. The \text{DataSizeVarThreshold} of (6) is used to determine whether the data sizes of compressed video frames are steady, increasing, or decreasing.

The third piece of data that influences the decision making is the target frame data size as determined by the current state of the cell phone network. From the estimated average frame data sizes and the percentage of delivered frames, MoStVid dynamically generates a target frame data size for future frames. The target frame data size (in bytes) is initialized to be the estimated average frame data size and is updated as follows:
\[ \text{TargetFrameDataSize} = \text{EstAvgFrameDataSize} \times \text{DeliveredFramePercent} \]

\[ \text{if } \text{fps} < \text{max fps} \text{ or } \text{DeliveredFramePercent} < 100\% \]

\[ \text{TargetFrameDataSize} = \text{TargetFrameDataSize} \times 1.05, \]

\[ \text{otherwise} \]

where the \text{EstAvgFrameDataSize} is computed in (5) and the \text{DeliveredFramePercent} is computed as 100 - \text{EstDropPercent} from (3). The idea is that MoStVid determines what size (in bytes) a video frame should be based on the current state of the network (as determined by the percentage of delivered frames from previous intervals) and the average frame data size. Further, if the video frames are being delivered at a rate of 100\%, and the current frame rate in fps is at the maximum, MoStVid takes advantage of this by increasing the target frame data size by 5\% in order to improve video frame quality and probe for more bandwidth. Given the current target frame data size, the video encoder can adjust the value of \( q \) to produce compressed video frames that fit more in line with the available bandwidth so that fewer frames may be dropped.

When streaming any type of live video the following conditions can occur: bandwidth may be relatively steady, increasing, or decreasing; video frame data sizes may be relatively steady, increasing, or decreasing; and the adaptive video encoder compresses video frames to estimate the provided target frame data size for the bandwidth. The data sizes of the compressed video frames may be on target, above target or below target. There are 27 possible combinations of the above three conditions.

Table 6.2 contains 27 decision making rules which cover these possible cases for the real-time adjustment of the fps rate of the video camera. When the percentage of dropped frames is
increasing (column 2), the framework makes a decision to reduce the fps rate by between 0.5 and 2.0 depending on the trend in the data sizes of the compressed video frames (column 3) and whether the video encoder can compress the video frames to meet the target frame data size (column 4). The first nine rules of Table 6.2 are used to adjust the camera’s fps rate when bandwidth is sensed to be decreasing. For a steady bandwidth, the camera’s fps rate is mostly maintained or increased by 0.5 when the frame drop rate is zero. The next set of nine rules address the possible combinations when bandwidth is steady. The last case is when bandwidth is sensed to be increasing (percentage of dropped frames is decreasing). For this case, nine more rules are used to either decrease or increase the camera’s fps rate by between 0.5 and 1.0, or maintain fps. More than 100 experiments of streaming real-time mobile video over real cell phone networks were performed to generate and fine-tune these 27 rules. Each rule occurred repeatedly during these numerous experiments and thus allowed for the proper testing of their final values.
### Table 6.2. MoStVid decision making rules

<table>
<thead>
<tr>
<th>Rule #</th>
<th>Percentage of Dropped Frames</th>
<th>Data Size of Frames</th>
<th>Encoded Size vs. Target Frame Size</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&gt; 5% or Increasing</td>
<td>Increasing</td>
<td>Above</td>
<td>Decrease fps by 2.0</td>
</tr>
<tr>
<td>2</td>
<td>&gt; 5% or Increasing</td>
<td>Increasing</td>
<td>On</td>
<td>Decrease fps by 1.0</td>
</tr>
<tr>
<td>3</td>
<td>&gt; 5% or Increasing</td>
<td>Increasing</td>
<td>Below</td>
<td>Decrease fps by 0.5</td>
</tr>
<tr>
<td>4</td>
<td>&gt; 5% or Increasing</td>
<td>Steady</td>
<td>Above</td>
<td>Decrease fps by 1.0</td>
</tr>
<tr>
<td>5</td>
<td>&gt; 5% or Increasing</td>
<td>Steady</td>
<td>On</td>
<td>Decrease fps by 1.0</td>
</tr>
<tr>
<td>6</td>
<td>&gt; 5% or Increasing</td>
<td>Steady</td>
<td>Below</td>
<td>Decrease fps by 0.5</td>
</tr>
<tr>
<td>7</td>
<td>&gt; 5% or Increasing</td>
<td>Decreasing</td>
<td>Above</td>
<td>Decrease fps by 1.0</td>
</tr>
<tr>
<td>8</td>
<td>&gt; 5% or Increasing</td>
<td>Decreasing</td>
<td>On</td>
<td>Decrease fps by 0.5</td>
</tr>
<tr>
<td>9</td>
<td>&gt; 5% or Increasing</td>
<td>Decreasing</td>
<td>Below</td>
<td>Decrease fps by 0.5</td>
</tr>
<tr>
<td>10</td>
<td>Steady</td>
<td>Increasing</td>
<td>Above</td>
<td>Decrease fps by 0.5</td>
</tr>
<tr>
<td>11</td>
<td>Steady</td>
<td>Increasing</td>
<td>On</td>
<td>Maintain</td>
</tr>
<tr>
<td>12</td>
<td>Steady</td>
<td>Increasing</td>
<td>Below</td>
<td>Maintain</td>
</tr>
<tr>
<td>13</td>
<td>Steady</td>
<td>Steady</td>
<td>Above</td>
<td>Increase fps by 0.5 if dropped frame rate is zero otherwise Maintain.</td>
</tr>
<tr>
<td>14</td>
<td>Steady</td>
<td>Steady</td>
<td>On</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Steady</td>
<td>Steady</td>
<td>Below</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Steady</td>
<td>Decreasing</td>
<td>Above</td>
<td>Maintain</td>
</tr>
<tr>
<td>17</td>
<td>Steady</td>
<td>Decreasing</td>
<td>On</td>
<td>Maintain</td>
</tr>
<tr>
<td>18</td>
<td>Steady</td>
<td>Decreasing</td>
<td>Below</td>
<td>Increase fps by 0.5</td>
</tr>
<tr>
<td>19</td>
<td>Decreasing</td>
<td>Increasing</td>
<td>Above</td>
<td>Decrease fps by 1.0</td>
</tr>
<tr>
<td>20</td>
<td>Decreasing</td>
<td>Increasing</td>
<td>On</td>
<td>Maintain</td>
</tr>
<tr>
<td>21</td>
<td>Decreasing</td>
<td>Increasing</td>
<td>Below</td>
<td>Maintain</td>
</tr>
<tr>
<td>22</td>
<td>Decreasing</td>
<td>Steady</td>
<td>Above</td>
<td>Maintain</td>
</tr>
<tr>
<td>23</td>
<td>Decreasing</td>
<td>Steady</td>
<td>On</td>
<td>Maintain</td>
</tr>
<tr>
<td>24</td>
<td>Decreasing</td>
<td>Steady</td>
<td>Below</td>
<td>Increase fps by 0.5</td>
</tr>
<tr>
<td>25</td>
<td>Decreasing</td>
<td>Decreasing</td>
<td>Above</td>
<td>Maintain</td>
</tr>
<tr>
<td>26</td>
<td>Decreasing</td>
<td>Decreasing</td>
<td>On</td>
<td>Increase fps by 0.5</td>
</tr>
<tr>
<td>27</td>
<td>Decreasing</td>
<td>Decreasing</td>
<td>Below</td>
<td>Increase fps by 1.0</td>
</tr>
</tbody>
</table>
6.4.3 Balancing Tradeoff between Video Compression and fps

If the fps rate drops too quickly (e.g., from 15 to 7), the result is that the video encoder will not receive video frames from the camera quickly enough for use in the heuristic to find a $q$ value that will generate compressed frame data sizes to match the target frame data size. Such a drastic change in the fps rate will result in the underutilization of the available bandwidth. Consequently, if the target frame data size increases too quickly, the video encoder will generate video frames that are too large for the available bandwidth. This will result in a large increase in the number of video frames that are dropped in the network.

A delicate balance must be maintained while adjusting the target frame data size (which controls the video compression level) and the fps rate to stream real-time video frames at data sizes and rates that fit the available bandwidth. Because the resulting data sizes of compressed video frames are dynamic and the bandwidth at any given time is also dynamic, to be consistently successful at delivering video frames, the decision making model of MoStVid must be very flexible to accommodate any type of video and any pattern or quality of bandwidth.

For a network that is stable, fewer frames should be lost. But if the average data size of video frames is increasing, aside from reducing the target frame data size using (7) (resulting in the video encoder altering $q$) to increase the video compression level, MoStVid will reduce the fps to adapt to this change. On the other hand, if the average data size of video frames is steady and the network is relatively stable, MoStVid will try to take advantage of this stability by increasing the target frame data size using (7) to improve the video frame quality. Similarly, an improving bandwidth warrants an increase in the fps sampling rate as well.
Fig. 6.5 plots a simplified but intuitive relationship between fps and compressed video frame data size in a limited bandwidth situation. The actual relationship between fps and compressed video frame data size at any given time is dynamic and dependent on the particular video being encoded, the available bandwidth, the frame rate in fps, and the speed of the processor on the mobile device. This dynamic and complex relationship is difficult to describe using conventional mathematical methods. However, MoStVid is able to effectively capture this relationship in its model. When bandwidth is limited, a tradeoff between fps and video compression level must be made to better utilize available bandwidth. Between the changes in video compression level through the target frame data size of (7), and the changes in fps sampling rate through the rules of Table 6.2, MoStVid adapts based on the changing situations of both the video being captured and the state of the network. The Experimental Results demonstrates how successful the two are and also demonstrates how dynamic the situation can be.

![Diagram of bandwidth utilization line and tradeoff between fps and compressed frame data size](image.png)

**Fig. 6.5.** Tradeoff between fps and video compression level. An illustration of a simplified tradeoff between frame per second rate and video compression level in a limited bandwidth situation.
6.4.4 Mobile–Network–Condition Awareness Features

From extensive experiments of streaming live video on three real cell phone networks (see Table 6.4), the following five naturally occurring cell phone bandwidth patterns were identified:

1. Steady (whether high, medium or low bandwidth)
2. Lightly fluctuating
3. Heavily fluctuating
4. Plummeting
5. Gradually decreasing/increasing

While these patterns are not necessarily visible while the video is being transmitted, the feedback provided in five-second intervals allows MoStVid to respond to each interval in micro-steps, and the resulting cumulative effect is that MoStVid adapts well in all conditions.

As stated in section 6.2, there are three conditions which can lead to packets being delayed or lost in mobile and wireless networks: network congestion, bit errors in wireless links, and mobility handoff. MoStVid is designed to infer and respond accordingly to these network conditions based upon the observed bandwidth. Network congestion can manifest itself in a variety of bandwidth patterns, including plummeting, low but steady, gradually decreasing, and possibly heavily fluctuating bandwidth. MoStVid responds to network congestion by decreasing bandwidth consumption by using (7) to increase the video compression level and rules #1-9 from Table 6.2 to reduce video transmission rate in fps.

Bit errors can occur randomly at any time in wireless links. This sometimes manifests as a pattern of lightly or heavily fluctuating bandwidth due to random packet corruption/loss. As a
result, MoStVid uses (3) and (4) to provide a smoother estimate of the available bandwidth for use in the decision making process. In doing so, MoStVid handles bit error-induced video frame losses by not overreacting or assuming that such video frame losses are caused by network congestion. Also notice that rules #1-9 from Table 6.2 are applied only if the percentage of dropped video frames is greater than 5%. MoStVid allows a 5% tolerance or an increasing frame drop rate threshold based on (4) before selecting any of these rules. This tolerance makes MoStVid robust to varying bandwidth due to bit errors, while also allowing it to quickly respond to dropping bandwidth that can be caused by network congestion.

Finally, a base station can decide to hand off an on-going call on a mobile device to another base station if the signal strength between it and the mobile device is fading or if the base station is overloaded with calls. Packets can be delayed or lost during a handoff [9]. Handoff induced frame loss and delay can manifest as gradually decreasing bandwidth due to fade in the signal strength or a steady then plummeting bandwidth which later increases again. MoStVid adapts to these bandwidth patterns by selecting appropriate rules from Table 6.2. For example rules #11-17, 20-23, and 25 are used to maintain video transmission rate in fps when the relationship between bandwidth and the compressed frame data size is sensed to be steady. Table 6.3 provides a summary of how MoStVid infers and adapts to changing network conditions based on changing bandwidth and video patterns.
6.4.5 Adaptations to Limited Mobile Computing Resources

Along with the considerations regarding available bandwidth and the level of video compression, other factors that need to be taken into account for the transmission of video over wireless networks include the limited computational capabilities and power constraints of the mobile device [38]. In addition to limited computing resources, mobile devices may dynamically reduce the processor speed to limit power consumption so as to conserve and extend battery life resulting in the mobile device running at less than its rated processor speed [15]. Because the processor speed of mobile devices is yet another variable which impacts the real-time streaming of video, MoStVid has several features that allow it to adapt to the limited and variable computing resources on mobile devices.

When processor speed is reduced, the video camera on the mobile device also captures video at a slower and less than optimal frame rate. For example, a mobile device that is capable of capturing video at a maximum frame rate of 15 fps may only capture video at a rate of 7.5 fps.
when the processor speed of the device is reduced. This in turn typically increases the video frame delivery rate because fewer frames are transmitted and therefore more frames are delivered. As processor speed drops and video frames are generated at a lower fps, MoStVid is able to utilize the additional bandwidth by increasing the video frame quality using (7). Notice from Table 6.2 that the adjustments of the fps sampling rates are made relative to the current rate at which the video camera can capture frames. MoStVid adapts well regardless of the starting/current frame rate of the video camera, the processor speed, or the power constraints of the mobile device.

A critical feature of MoStVid is that it avoids the direct measurement of bandwidth which can be computationally expensive and can place additional load on the network. The available bandwidth is predicted using minimal feedback from the video decoder and the dynamic data size of the particular video being streamed. This information is already available on the mobile device. The bandwidth estimation model is therefore computationally simple, efficient, and well suited for the computing resources available on mobile devices.

Finally, the video encoder uses 5-3 DWT for transforming the raw video frames, as discussed in chapter 4. The 5-3 wavelet was selected because it is computationally feasible for mobile devices. It is very scalable, producing a graceful degradation of video quality at low bit rates and providing better compression than the discrete cosine transform used in JPEG [52]. The 5-3 DWT performs integer to integer transforms and can be implemented using additions and bit shifts, avoiding multiplications [57]. The avoidance of wavelet filters with floating point transforms is particularly important for the real-time streaming of video on mobile devices. Floating point computations are very slow on mobile devices [51], consume more energy, and may violate the real-time requirements of the video compression process. While floating point
wavelet filters offer better compression, their advantage over integer wavelet filters is marginal [10]. Although the integer 5-3 DWT can be replaced with other wavelet filters if desired, MoStVid uses the 5-3 DWT because it permits the real-time compression of video on the relatively weak processing power of the cell phone CPU.

Some desktop video compression algorithms employ inter-frame compression to reduce the temporal redundancy in videos. Preliminary research of incorporating inter-frame compression into the 5-3 wavelet video encoder shows that this approach can place unacceptable computational demands on mobile devices [13]. Inter-frame compression requires additional memory and complex computation for determining the differences between video frames, implementing a necessary video decoder as part of the video encoding process, or computing 3D wavelet transform across multiple frames [62], [50], [69], [70]. In addition, the dropping of packets is always an issue given the error-prone nature of cell phone networks and the best-effort Internet. This situation is exacerbated by the fact that the loss of packets in a reference frame will result in a significant decrease in the video quality as errors persist across all subsequent dependent frames until another reference frame is successfully received. As stated in [71], a 3% packet loss in the Internet can translate into 30% frame error and a 20% packet loss can multiply to over 90% frame error. Given these factors, inter-frame compression is not practical for streaming video in real-time over existing cell phone networks.

It should be noted that any suitable video encoder can be used with the MoStVid framework. The MoStVid framework is loosely coupled with the wavelet video encoder. Fig. 6.4 shows the MoStVid decision engine communicating only the target frame data size to the wavelet video encoder. The framework only demands that the video encoder be able to adapt the data size of the compressed video frames to approximate the target frame data size provided in bytes. As a
result, the wavelet video encoder can easily be replaced with other emerging video compression techniques such as a compressed sensing-based video encoding scheme [72].

### 6.5 Experimental Setup

In chapter 4, experiments showed that the wavelet transformation and the video encoding permits real-time streaming video over cell phone networks. In chapter 5, it was noted that in spite of the reduced data size of the videos transmitted, cell phone networks drop numerous frames because of limited and variable bandwidth. This chapter highlights that the enhanced version of MoStVid, by reactively adapting to the available network bandwidth, can improve the percentage of frames delivered from below 50% to close to 100%.

To investigate the real-time adaptability of MoStVid, numerous live video experiments were performed over the three types of cell phone networks, evaluating video quality and the frame delivery rate. The three cell phone networks are the 2.5G *Enhanced Data rates for GSM Evolution (EDGE)* network with a maximum rated bandwidth of 120 kbit/s (but typical rates are around 30 kbit/s [32]), and two 3G networks: *High Speed Packet Access (HSPA)* and *Evolution-Data Optimized Rev. A (EVDO)* with maximum rated bandwidth of up to 14 Mbit/s and 3.1 Mbit/s respectively [32]. 3G network coverage and bandwidth can be patchy even in major cities, with actual measured throughput typically ranging between 50 kbit/s and 300 kbit/s for HSPA, and about 90 kbit/s for EVDO [32]. Table 6.4 gives a summary of the networks used in the experiments.
The 27 rules in Table 6.2 were developed with over 100 live streaming video experiments which were performed on the three different cell phone networks listed in Table 6.4. The rules were fine-tuned to their final values over the course of the more than 100 experiments. These 27 rules occurred repeatedly during the experiments, providing ample opportunity for each rule to be thoroughly tested and fine-tuned. After the fine-tuning experiments, the rules were fixed. Once these rules were developed, an additional 60 experiments were performed. This chapter reports on these last 60 experiments, 20 on each network. The experiments were performed over a period of six weeks at different hours of the day, during weekdays, weekends, and holidays in order to capture the general behavior of the networks. Three test cell phones were used, one for each network, each having 500 MHz processor speed, 96 MB RAM for EDGE, 107 MB RAM for HSPA, and 114 MB RAM for EVDO. On fresh reboot, due to other software running on the cell phones, each cell phone had lower available memory for video compression and streaming: 61 MB, 63 MB, and 60 MB, respectively.

Each experiment lasted between 3 and 15 minutes, and transmitted between 845 and 8500 video frames. Each raw video frame was 240x320 pixels at 8 bits per pixel. The experiments were conducted using live video of outdoor scenes, including complex landscapes featuring

### Table 6.4.
Profile of cell phone networks

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Rated Bandwidth</th>
<th>Typical Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5G – EDGE</td>
<td>Up to 120 kbit/s</td>
<td>30 kbit/s</td>
</tr>
<tr>
<td>3G – HSPA</td>
<td>Up to 14 Mbit/s</td>
<td>50 kbit/s – 300 kbit/s</td>
</tr>
<tr>
<td>3G – EVDO Rev. A</td>
<td>Up to 3.1 Mbit/s</td>
<td>90 kbit/s</td>
</tr>
</tbody>
</table>
trees, shrubs, grass, rocks, and running water, and indoor scenes with textured backgrounds, while avoiding videos with un-textured backgrounds such as plain walls. Videos with textured background like outdoor scenes are challenging to compress, but also represent the sorts of video that cell phone users are likely to capture.

The video camera and wavelet video encoder running on the cell phones is capable of encoding video sequences at frame rates ranging from 1 fps to 15 fps. However, for these experiments a minimum frame rate of 3 fps was utilized. In each experiment, the frame rate started at 15 fps and \( q \) was initialized to 12 in the video encoder. The evaluation of the resulting video quality in these experiments is made relative to the fact that the cell phones can achieve a maximum of 15 fps. As such, smoothly delivered fps rates above 10 fps are considered very good quality video when combined with video frames of good or better quality. At five-second intervals during the course of each experiment, MoStVid dynamically adjusted the frame rate in fps and target frame data size to better utilize the available bandwidth and computing resources. A five-second interval was chosen because the limited computing resources of the mobile devices presented a bottle neck in the form of a noticeable pause in the video when feedback (of the number of frames delivered) was sent more frequently. If feedback is lost, MoStVid uses the previously received feedback for decision making for the five-second interval. The decision making process and video adaptations is robust enough to handle lost feedback.

Table 6.5 shows a breakdown of how each of the three cell phone networks performed with respect to the five bandwidth patterns identified in section 6.4.4. This table shows a summary of bandwidth patterns occurring in the 60 experiments that were performed on the cell phone networks.
6.6 Experimental Results

Figs. 6.6-6.11 show representative examples of real-time mobile streaming video demonstrating the bandwidth patterns. In each figure, three graphs are provided, denoted as (a), (b) and (c). Graph (a) plots the frames per second transmitted versus delivered, graph (b) illustrates the percentage of frames delivered, and graph (c) shows the target frame data size versus the average frame data size transmitted.

Fig. 6.6 shows experimental results from a live streaming video that occurred during a period of heavy fluctuation in bandwidth. MoStVid was disabled for the first 55 seconds of the experiment to show what can happen without an adaptive framework. In the first 55 seconds of Fig. 6.6a, notice the large disparity between the transmitted fps and the delivered fps, with the delivered fps being as low as 2.5 fps. The first 55 seconds of Fig. 6.6b show the corresponding heavy fluctuation in the percentage of frames delivered with frame delivery rates as low as 16.7%. During the same 55 second period, the average value for $q$ was 12 (excellent video frame quality).

### Table 6.5.
Summary of bandwidth patterns occurring in 60 experiments on networks

<table>
<thead>
<tr>
<th>Bandwidth Pattern</th>
<th>EDGE</th>
<th>HSPA</th>
<th>EVDO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steady (high or low)</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Lightly fluctuating</td>
<td>5</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Heavily fluctuating</td>
<td>10</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Plummeting</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Gradually decreasing/increasing</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>


Fig. 6.6. Heavily fluctuating bandwidth video stream. MoStVid was disabled for the first 55 seconds and enabled for the remainder of the video. Average values from 5s-55s are $q=12$ (excellent quality), delivered fps=7.5, frame delivery=50%, and resulting video quality=poor. Average values from 60s-360s are $q=19.7$ (high quality), delivered fps=11.9, frame delivery=98.5%, and resulting video quality=high. (a) Transmitted fps vs. Delivered fps. (b) Percentage of delivered frames. (c) Target frame data size vs. Average frame data size.

quality), average delivered fps was 7.5, and the percentage of delivered frames was 50%. During the first 55 second time period, although each video frame quality was excellent, the resulting video quality was poor due to the randomly delivered fps rate of only 50%.

After 55 seconds MoStVid was enabled. Notice the improvement in the overall quality of the rest of the video stream as MoStVid adjusts the fps rate (in Fig. 6.6a) and target frame data size.
(in Fig. 6.6c) to be more aligned with the bandwidth. The remainder of the video improved, resulting in a 98.5% frame delivery, with an average $q$ value of 19.7 (high video frame quality) and an average delivered fps rate of 11.9.

Figs. 6.7-6.11 show the experimental results for live streaming video transmissions that occurred during each of the other four bandwidth patterns. In these experiments, unlike the experiment from Fig. 6.6, MoStVid was running throughout the time period. Fig. 6.7 illustrates an experiment in which there was steady bandwidth and therefore MoStVid made few changes to fps. Fig. 6.8 shows the results of an experiment from a lightly fluctuating bandwidth pattern in which, again, MoStVid did not need to make many changes to the fps rate, but the target frame data size was reduced to fit the bandwidth. Fig. 6.9, on the other hand, demonstrates a more extreme case of plummeting bandwidth. In this case, MoStVid had to make numerous changes in order to maintain the video delivery rate. Figs. 6.10 and 6.11 show cases of gradually decreasing and gradually increasing bandwidth. In these figures MoStVid gradually decreased or increased the fps rate in response to the gradual drop or increase in the rate of frames delivered at various points in the video streams.
Fig. 6.7. Steady bandwidth video stream with average values $q=21$ (high quality), delivered fps=11.7, frame delivery=98.8%, and resulting video quality=very good. (a) Transmitted fps vs. Delivered fps. (b) Percentage of delivered frames. (c) Target frame data size vs. Average frame data size.
**Fig. 6.8.** Lightly fluctuating bandwidth video stream with average values $q=19$ (high quality), delivered fps=14.7, frame delivery=98.9%, and resulting video quality=high. (a) Transmitted fps vs. Delivered fps. (b) Percentage of delivered frames. (c) Target frame data size vs. Average frame data size.
Fig. 6.9. Plummeting bandwidth video stream with average values from 5s-70s \( q=12.0 \) (excellent quality), delivered fps=14.6, frame delivery=99.5\%, resulting video quality=very high; from 75s-150s \( q=30.6 \) (low quality), delivered fps=6.9, frame delivery=80.2\%, resulting video quality=poor; and from 155s-210s \( q=27 \) (passable quality), delivered fps=3.0, frame delivery=97.9\%, resulting video quality=acceptable. (a) Transmitted fps vs. Delivered fps. (b) Percentage of delivered frames. (c) Target frame data size vs. Average frame data size.
Fig. 6.10. Gradually decreasing bandwidth video stream with average values $q=17.4$ (high quality), delivered fps=12.4, frame delivery=98.9%, and resulting video quality=high. (a) Transmitted fps vs. Delivered fps. (b) Percentage of delivered frames. (c) Target frame data size vs. Average frame data size.
Fig. 6.11. Gradually increasing bandwidth video stream with average values $q=23.4$ (good quality), delivered fps=14.1, frame delivery=95.3%, and resulting video quality=very good. (a) Transmitted fps vs. Delivered fps. (b) Percentage of delivered frames. (c) Target frame data size vs. Average frame data size.

For each of the bandwidth patterns, MoStVid was able to achieve a delivery rate close to 100% despite the changing bandwidth and changing video frame data size. These results demonstrate that MoStVid is capable of modeling the state of the cell phone network and through the rules of Table 6.2 and the target frame data size of (7), adapt appropriately. Even when bandwidth plummets, MoStVid is able to track the rapidly declining bandwidth by adjusting the target frame data size and the transmitted fps downwards such that the delivered fps are aligned with the plummeting bandwidth as seen in Fig. 6.9a.
Tables 6.6-6.8 show the best case, overall average, and worst case results of the five bandwidth patterns for the 60 experiments. In the best cases, shown in Table 6.6, MoStVid delivers high quality video at high fps rates with close to 100% of the frames delivered. For the case of plummeting bandwidth, MoStVid initially delivered very high quality video, followed by poor quality video during the period of plummeting bandwidth, and later stabilized to a smooth and acceptable quality video despite low bandwidth. This case is illustrated in Fig. 6.9. Table 6.7 shows the average values over all 60 video experiments. Here the resulting video quality was good or very good in all cases except for that of plummeting bandwidth, in which case video quality was still acceptable. Except for the case of plummeting bandwidth MoStVid achieved average delivery rates above 91% (well above 91% in most cases). Even among the worst cases (Table 6.8), MoStVid provided at least an 85% delivery rate and acceptable video quality (with the exception of plummeting bandwidth).

<table>
<thead>
<tr>
<th>Bandwidth Pattern</th>
<th>$q$ (Quality)</th>
<th>fps</th>
<th>% Delivered</th>
<th>Resulting Video Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steady (high)</td>
<td>17.90 (high)</td>
<td>13.10</td>
<td>98.50</td>
<td>High</td>
</tr>
<tr>
<td>Lightly fluctuating</td>
<td>14.70 (excellent)</td>
<td>14.45</td>
<td>99.40</td>
<td>Very high</td>
</tr>
<tr>
<td>Heavily fluctuating</td>
<td>17.70 (high)</td>
<td>12.96</td>
<td>98.70</td>
<td>High</td>
</tr>
<tr>
<td>Plummeting</td>
<td>23.40 (good)</td>
<td>8.30</td>
<td>91.73</td>
<td>Very high, poor, then acceptable</td>
</tr>
<tr>
<td>Gradually decreasing/increasing</td>
<td>17.40 (high)</td>
<td>12.40</td>
<td>98.90</td>
<td>High</td>
</tr>
</tbody>
</table>
Table 6.7.
Overall average values for q, fps, and percentage of delivered frames

<table>
<thead>
<tr>
<th>Bandwidth Pattern</th>
<th>q (Quality)</th>
<th>fps</th>
<th>% Delivered</th>
<th>Resulting Video Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steady</td>
<td>22.79 (good)</td>
<td>12.79</td>
<td>97.86</td>
<td>Very good</td>
</tr>
<tr>
<td>Lightly fluctuating</td>
<td>23.21 (good)</td>
<td>12.15</td>
<td>96.02</td>
<td>Very good</td>
</tr>
<tr>
<td>Heavily fluctuating</td>
<td>27.39 (passable)</td>
<td>9.36</td>
<td>91.98</td>
<td>Good</td>
</tr>
<tr>
<td>Plummeting</td>
<td>33.32 (low)</td>
<td>4.60</td>
<td>87.65</td>
<td>Acceptable</td>
</tr>
<tr>
<td>Gradually decreasing/increasing</td>
<td>22.33 (good)</td>
<td>11.13</td>
<td>95.92</td>
<td>Very good</td>
</tr>
</tbody>
</table>

Table 6.8.
Worst case average values for q, fps, and percentage of delivered frames

<table>
<thead>
<tr>
<th>Bandwidth Pattern</th>
<th>q (Quality)</th>
<th>fps</th>
<th>% Delivered</th>
<th>Resulting Video Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steady (low)</td>
<td>32.50 (low)</td>
<td>11.50</td>
<td>96.50</td>
<td>Acceptable</td>
</tr>
<tr>
<td>Lightly fluctuating</td>
<td>33.10 (low)</td>
<td>13.14</td>
<td>95.50</td>
<td>Acceptable</td>
</tr>
<tr>
<td>Heavily fluctuating</td>
<td>34.10 (low)</td>
<td>8.68</td>
<td>85.60</td>
<td>Acceptable</td>
</tr>
<tr>
<td>Plummeting</td>
<td>34.92 (low)</td>
<td>3.60</td>
<td>85.98</td>
<td>Poor due to excessively low bandwidth</td>
</tr>
<tr>
<td>Gradually decreasing/increasing</td>
<td>25.60 (passable)</td>
<td>10.19</td>
<td>94.10</td>
<td>Good</td>
</tr>
</tbody>
</table>

In all 60 experiments, the streamed videos were challenging to compress. The worst case videos represent a particularly daunting challenge in that, because of the complexity of the content of these videos, the video encoder could not compress the video at the target frame data size for the bandwidth. These worst case videos were compressed at frame data sizes that were of lower quality, but above the target frame data sizes for the available bandwidth at the time.
These videos illustrate that depending on the content of the video being streamed, video quality can be compromised in a limited and variable bandwidth situation, although again MoStVid succeeds in delivering the majority (over 85%) of the video frames. For the worst case of plummeting bandwidth, the resulting video quality was poor because the available bandwidth dropped to the point where the particular video being streamed could not be compressed further (given the lowest compression limit of $q=35$) to meet the target frame data size. Whenever the compressed video frame data size is above the target frame data size (column 4) in the rules of Table 6.2, MoStVid handles these difficult to compress videos by either reducing or maintaining the fps rate to deliver a high percentage of the frames. In general, although the worst case videos were encoded at a low quality, the high delivery rate achieved by MoStVid nonetheless resulted in useful video with good or acceptable quality.

6.7 Summary

This chapter has presented a careful examination of the challenges which are unique to the streaming of real-time mobile video over cell phone networks. In particular, these challenges involve three interacting sets of constraints: variations in bandwidth, variations in video frame data size, and variations in the computing resources (CPU, memory, battery power) available to mobile devices. In numerous experiments, live video originating from cell phones was streamed over real cell phone networks, addressing the dynamic nature of actual bandwidth, and also identifying a set of bandwidth patterns that must be taken into account if real-time video is to be delivered successfully.
This chapter provides a thorough description of the final version of MoStVid, a real-time application layer solution to the problem of streaming live video originating from mobile devices. MoStVid is an adaptive framework for real-time mobile streaming video that produces significant improvement on bandwidth usage, enabling a high delivery rate of useful video. Experimental results show that without an adaptive system, video frame delivery rates can be less than 50%. With MoStVid, video frame delivery rates are significantly improved so that, in the best cases, MoStVid is able to offer high quality real-time mobile video with delivery rates of close to 100%. In the average cases, MoStVid offers very good quality video with delivery rates above 91%, and in the worst cases it offers useful video with delivery rates of above 85%. Building upon the high video frame delivery rate achieved by MoStVid, future work will incorporate loss concealment schemes in the video decoder to hide the visual effects of the few frames lost in the network in order to further improve the perceived video quality.

As predicted in [38], a major hurdle for future generations of wireless networks is that the profusion of complex and feature-rich multimedia applications will pose substantial challenges to the capacity of the network to deliver them. As such, irrespective of the rated bandwidth and given that no wireless network can provide equal quality service in every physical location, an adaptive framework for real-time streaming video is desirable.
Chapter 7  Future Research Directions

The streaming of real-time video from the cameras of mobile devices is expected to have a significant impact in a number of settings, including economic development, law enforcement, emergency medical response, and education. The problem of streaming real-time video from mobile devices is a new and untapped area of research. This topic has only minimally been explored, because mobile devices are now only attaining a level of maturity that can support very complex tasks such as the streaming of video from the devices. However, a number of the problems discussed in this dissertation have already been addressed in the context of desktop computers and video streaming over stable and rich bandwidth environments. Video compression algorithms contain many complex components that have themselves been developed by experts specializing in very specific aspects of the algorithm. For example, experts on 3D wavelets tend to only work on their specialization.

One challenge is to select and scale the components of existing video compression algorithms that are most suitable to the computational resources available on mobile devices. Because of the huge body of research spanning multiple fields which must be engaged, new researchers who are interested in the challenges of streaming video in real time from mobile devices should pick a specific area of interest and investigate how to scale it to operate on mobile devices. For example, the motion estimation component of MPEG-4 Encoders can be computationally intensive, even for desktops. Research resulting in the ability of this component
to function efficiently on mobile devices in real time would represent a significant breakthrough with diverse applications.

Another direction of research is in the area of energy consumption. It has already been established that more computation results in greater energy consumption. However, it is not clear how the various components of video compression algorithms influence the battery life of mobile devices. Investigations identifying the energy efficiency of various components of these video compression algorithms would be very useful and could identify which components should be preserved and which eliminated due to excessive energy consumption. More specifically, research could be undertaken to examine the tradeoff between the contributions of each component compared to its energy demands.

Future research should seek to exert greater control of the real-time video being streamed over cell phone networks from the cameras of mobile devices. In addition to simulations in labs, research should be conducted on real mobile devices so that both the constraints of the mobile devices and the cell phone networks can be taken into account while scaling video compression algorithms to work within the real-time constraints of mobile devices and wireless networks.

Further, if the situation permits, the user preferences of higher fps rate or higher quality video frames could be taken into account in order to adapt the real-time video to better suit the particular desires of the user. In certain situations, adapting the video may require taking into consideration privacy-related issues such as the blanking of background or foreground images to obscure unnecessary content. Many other possibilities exist for video adaptation.

The availability and usage of memory on mobile devices is also a fruitful area of research. Mobile device memory is limited and varies greatly between devices, and can vary from moment
to moment depending on the demands of the operating system and other essential software. Research could be conducted to examine how much memory is available to the video compression algorithm for real time video compression and streaming and what type of adaptations are feasible on a given device. For example, if inter-frame compression is being employed, how many frames can be in a GoP at the time of video streaming?

Another area of potential innovation is the development of realistic simulations of cell phone network bandwidth. Because the bandwidth of wireless networks can vary significantly, as described in section 3.2, researchers must make additional efforts to simulate correctly the conditions found in real world networks. The accurate modeling of wireless bandwidth behaviors would greatly facilitate the ability of researchers to test a variety of software and hardware issues in the lab.

Mobile phone processors themselves are targets for research. With limited computational capabilities, further benchmark tests should be run to determine their capabilities under a number of constraints. Power management systems, while critical to the function of these devices, can nonetheless alter processor power when the battery begins to drain. This topic should be examined to better understand how the reduction in processor speed can impact the performance of real time video compression algorithms and the streaming of video. Such information could lead to methods to mitigate the degradation of computational capacity from necessary power management functions.
The big secret in life is that there is no big secret. Whatever your goal, you can get there if you're willing to work.

- Oprah Winfrey

Chapter 8  Conclusions

8.1 Contribution of Research

This dissertation contributes the following:

- A foundation for future researchers through the identification and thorough analysis of the constraints and challenges of compressing and streaming real-time video originating from mobile devices. This examination identified and took into consideration,
  - the limited computing resources of the mobile devices;
  - the impact of the limited and variable battery power of mobile devices;
  - the nature of the real-time video to be compressed and streamed from the camera of mobile devices;
  - the variable and limited bandwidth of cell phone networks, and the impact of these constraints on the streaming of real-time video;
  - and the computational influence of various components of video compression algorithms.

- From an understanding of the constraints and challenges involved, an adaptive solution was developed to solve the problem.
• An efficient real-time video compression algorithm to compress video from the cameras of mobile devices to attain the camera’s maximum frames per second (fps) rate.

• An adaptive real-time bandwidth estimation model to estimate the available bandwidth of the cell phone network at the time of the video streaming in order to achieve high delivery rate of the video frames.

• An implementation of a real-time video streaming framework that uses the developed real-time video compression algorithm and bandwidth estimation model to compress and stream live video originating from real mobile phones over existing cell phone networks.

• Because the behavior of real cell phone network bandwidth have not been studied in detailed, the results of this research can be used to develop a performance matrix for examining and evaluating the quality of the bandwidth of real world cell phone networks.

8.2 Concluding Remarks

Along with the development and implementation of an adaptive solution, this dissertation describes the constraints, challenges, and the benefits of streaming real-time video from mobile devices across existing cell phone networks. While much of the research to date has concentrated on streaming video from computer to mobile device or from computer to computer over the Internet, this research establishes a foundation for researchers who want to address the significantly harder problem of streaming real-time video from mobile devices. This dissertation provides a solution, and identifies and analyzes the various problems that are specific to streaming real-time video from mobile devices over cell phone networks. The chapter on future
research directions describes numerous research problems and opportunities that are yet to be explored.

The solution to compressing and streaming real-time video from mobile phones over cell phone networks must integrate elements from a number of normally discrete fields of computer science and engineering, particularly wireless networking, mobile computing, data compression, signal processing, artificial intelligence, and real-time systems. The completion of this interdisciplinary research via the methodology of using actual mobile devices, real-time video, and actual cell phone networks is unique, and shows that successful and comprehensive research can be conducted with this approach.

There are numerous open questions that continue to present challenges. Some of these questions include: Will a 4G network have more stable and higher bandwidth with features to support real-time mobile video? Will the multimedia demand on the cell phone network increase with improved bandwidth? When will mobile phone processors reach a capability equivalent to modern desktop computers, for instance by employing superscalar pipelines? Will there be additional breakthroughs in compression technologies to better support streaming real-time video under such tight conditions? Questions such as these highlight the fact that research into the streaming of real-time video from mobile devices will only grow in importance in the years to come.
References


REFERENCES


http://trace.eas.asu.edu/yuv/qcif.html

http://media.xiph.org/video/derf/


A Brief Biography

Ngozi V. Uti received a BS *summa cum laude* in Computer Science from Northern Kentucky University (NKU) in 2002 and an MS in Computer Science from Indiana University (IU), in 2004. She is currently working towards the completion of her PhD in Computer Science and Engineering at the University of Cincinnati. From 2005-2007 she was a Senior Software Engineer at the Cincinnati Financial Corporation. She was an Associate Instructor in the Department of Computer Science at IU from 2002-2004 and an Instructor in the Department of Computer Science at NKU from 2008-2009. She was also the Co-Director of the annual NKU Computer Science Scholars Summer Camp for 2009 and 2010. In 2001 she received the Best Student Paper Award from the Consortium for Computing in Small Colleges at the Midwestern Conference. She was awarded the Outstanding Senior in Computer Science Award at NKU in 2003. She is an NSF Graduate Research Fellow and a Ford Pre-Doctoral Fellow for her 2007 awards of the National Science Foundation Graduate Research Fellowship and the Ford Foundation Pre-Doctoral Fellowship, both for her doctoral research. Her research interests include Mobile Computing, Artificial Intelligence, and Real-Time Mobile Video Compression, Analysis, and Streaming over Cell Phone Networks. She is a student member of the IEEE, IEEE Women in Engineering, and the IEEE Computer Society.
B List of Publications

B.1 Invited Book Chapter from this Dissertation


B.2 Conference Proceedings from this Dissertation


B.3 Journal Article in Progress from this Dissertation

B.4 Other Publications


The above journal article is a revised and augmented version of SIGCSE 2002, selected by IMEJ editors as among the conference's best contributions in educational technology.