UDAB-ViS: User Driven Adaptable Bandwidth Video System

Dustin Wright and Yusuf Ozturk

March 31, 2014

Abstract

Adaptive bitrate (ABR) streaming has become an important and prevalent feature in many multimedia delivery systems, with content providers such as Netflix and Amazon using ABR streaming to increase bandwidth efficiency and provide the maximum user experience when channel conditions are not ideal. Where such systems could see improvement is in the delivery of live video with a closed loop cognitive control of video encoding. In this research, we present a camera system which provides spatially and temporally adaptive video streams, learning the users preferences in order to make intelligent scaling decisions. The system employs a mobile phone (Android) application serving as a live video streaming server with a hardware based H.264/AVC encoder for video compression. The encoding parameters can be configured by the user or by the cognitive system on behalf of the user when the bandwidth changes. The cognitive video client developed in this study learns the users preferences (i.e. video size over frame rate) over time and intelligently adapts encoding parameters when the channel conditions change. It has been demonstrated that the cognitive decision system developed has the ability to control video bandwidth by altering the spatial and temporal resolution, as well as the ability to make scaling decisions for the user when a sufficient amount to data has been collected.

1 Introduction

As wireless networks become more ubiquitous and the number of devices capable of accessing these networks increases, the need for more efficient video streaming solutions becomes vastly important. By 2015, it is expected that approximately 90 percent of online consumer traffic will be video and almost 66 percent of mobile traffic will be video [7]. With increasing amount of video traffic, bandwidth efficiency becomes a serious concern in order to deliver the best quality of experience (QoE) to each individual user. At the same time, servers should be able to deliver video in such a way that information the user deems important is not lost due to bandwidth constraints and the method by which the video adapts.

To date, numerous solutions have been developed that take a range of approaches to solve this problem. Many of these approaches are based on already existing protocols such as the Real-Time Streaming Protocol RTSP [9], TCP, and HTTP [7, 16, 17, 13] and achieve bandwidth adaptability in one of a few ways. These bandwidth adaptability techniques include progressive download streaming, adaptive bitrate (ABR) streaming, and stream-switching [5]. In progressive download streaming, video is transmitted as regular data files using TCP and is buffered by the client; playing starts when a sufficient amount of buffering has been achieved. This technique is employed by the popular video streaming website YouTube. Other solutions such as Netflix and Amazon Video employ ABR, in which the server can select the encoding bitrate in order to maximize the video quality for a given channel. The result is a drastic reduction in the need for buffering online video consumption; from the end user's perspective, the quality resolution of the video will change as network conditions change. Finally, with stream switching, the server encodes the source video with different encoding parameters and allows the client to switch between streams based on network conditions. Examples of stream switching solutions can be found in [5],[16] and [17]. Such techniques are ideal for online video streaming as they can use HTTP to negotiate streaming parameters and transmit the video stream; however, the major pitfall is that in most cases, only the video bitrate will be affected and no control is exercised over the spatial and temporal resolution of the video. In the case where spatial and temporal resolution can be affected, raw video will have to be re-encoded or transcoded at the source which can cause a delay in the video being transmitted [5].

In addition to protocol based approaches which tend to only allow for bitrate scalability, many codec based approaches have been developed which allow for easy spatial and temporal scalability [6]. Two prime examples of codec based approaches are H.264/AVC and H.264/SVC [2][4]. With these codecs, video need only be encoded once and, due to the nature of the decoders, can be transmitted as several sub-streams in order to control the temporal or spatial resolution. The primary issue with this approach is that it limits the client to only a certain set of video decoders. A review of the H.264/SVC approach to scalability will be presented later in this paper.

A problem with many of these solutions is that only quality resolution is affected in order to control the video bandwidth, which may or may not be acceptable depending on the user or the application. Numerous popularly used streaming solutions such as HTTP Adaptive Streaming (HAS) [7] and Dynamic Adaptive Streaming over HTTP (DASH) [10], will throttle the bitrate of a video stream in order to adapt to changes in channel bandwidth, which will result in a change in video quality. This behavior is exhibited by the widely used video content provider Netflix, which employs DASH (and thus ABR) to achieve bandwidth adaptability [13]. A prime example where this system will not be viable is in medical teleconferencing. A patient may wish to have a remote consultation with his or her doctor in order to quickly and efficiently receive feedback about a medical problem such as a burn or lesion they may have endured. Were the video bandwidth to be controlled by ABR and the quality resolution be affected in order to fit to the channel, the doctor may not be able to properly diagnose or provide valid feedback to the user because the visual quality is not sufficient. However, if the system were to scale the temporal resolution of the video instead of the quality, visual information would be kept intact and the less important temporal information would be sacrificed. The system we developed uses this kind of scalability in tandem with the generation of user profiles in order to make an intelligent determination of how to alter each resolution. In this approach, video context is taken into consideration to provide different video scalability options in different situations. This addresses the issue of how to control the different encoding parameters in such a way as to align with the users desires without requiring the user to change the parameters themself.

One of the key factors to achieve such a system is the automation of video bandwidth control. In [5], the authors propose a *Quality Adaptation Controller* which uses a proportional integrator (PI) controller at the server to select an appropriate video stream for the client. The system uses stream switching and employs feedback control to maintain the video bandwidth below the available channel bandwidth. This system succeeds in being a codec independent solution that is able to keep the video bandwidth at or below the available bandwidth with a transient time of less than 30 seconds, and is able to share the channel with concurrent streams [5]. However, the solution does not take into account the context of the video and the users desires in order to select encoding parameters in an intelligent fashion. In this study, a learning module using support vector machines (SVM) has been created which learns the users preferences and is able to adapt the video appropriately. These preferences are learned based on contextual video features such as channel bandwidth and video content type, thus allowing for different functionality in different situations.

The rest of the paper is structured as follows; in section 2 we review H.264 packetization; in section 3, we review H.264/SVC as a scalable streaming solution, as well as the scalability solution used in our system; sections 4 and 5 will describe the system architecture proposed in this study, as well as the method by which the video is encoded on the video server developed in this study; section 6 will detail our learning model and how video bandwidth is optimized; and finally, section 7 will present the test setup and experimental results.

2 Packetization and Frame Encoding for H.264

2.1 H.264/AVC Basics

H.264/AVC is a video coding standard developed jointly by the ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Moving Pictures Expert Group (MPEG) designed with the goals of enhanced video compression and "network friendly" video representation addressing "conversational" applications such as video conferencing, as well as "non-conversational" applications such as broadcast streaming [12]. In December of 2001 VCEG and MPEG formed a Joint Video Team (JVT) which in March of 2003 finalized the draft of the H.264/AVC video coding standard for formal submission [12].

The standard provides highly efficient video coding and can be used in a breadth of applications, from storage to streaming. It provides bitrate savings of 50% or more over its predecessor video codecs [15]. This makes H.264/AVC especially applicable in wireless environments. In [14] and [11], the authors review and show the effectiveness of H.264/AVC as a tool for video compression over IP networks and in wireless settings. In addition, H.264 employs a litany of features to enhance the quality and customization of video coding. Some examples of these features are

- Motion vectors over picture boundaries
- Multiple reference picture motion compensation: P-frames and B-frames can select from a large number of stored reference pictures to determine the values in the current frame [12]
- Weighted prediction

In addition, H.264/AVC supports a number of features to allow greater error resilience and flexibility over many environments [12]. Some of these important features are

- NAL unit syntax structure
- Flexible slice size
- Flexible Macroblock Ordering (FMO)
- Arbitrary Slice Ordering

[12] and [3] provide a detailed overview of each of these features. The structure of an H.264/AVC encoder is depicted in Figure 1. The codec given in

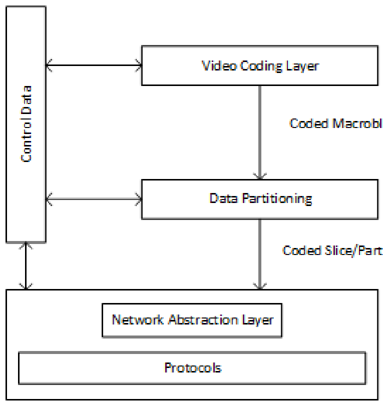


Figure 1: H.264/AVC Encoder Structure [12]

Figure 1 covers both the Video Coding Layer (VCL) and the Network Abstraction Layer (NAL). The VCL performs the physical encoding and compression of video while the NAL adds a header to video packet data and is exceptionally important to assist the decoder in understanding how to handle the packetized frames. We will now discuss the NAL and give a general overview of how frames are packetized when sent over UDP/RTP.

2.2 NAL Unit Headers and Packetization

The NAL allows the ability to map and packetize data to a multitude of transport layer protocols (i.e. UDP/RTP, file formats, etc.). The most important aspect that we will examine is the NAL unit and its relation to different types of transport layer payloads. In particular, we will look at the NAL unit when streaming video over UDP/RTP. When frames are encoded in the VCL they are organized into NAL units which serve as a wrapper to the data. Each NAL unit will contain a header byte that will indicate what type of data is contained in this unit. When streaming video using UDP/RTP in the transport layer, the beginning of each packet will contain a NAL unit. The NAL unit conveys relevant information for the decoder about the encoded data. It contains a one byte header and a payload byte string [15]. The header indicates the type of NAL unit, potential presence of errors, and information about the relative importance of this NAL unit in the decoding process [15]. The structure of the one-byte NAL unit header is shown in Figure 2. The fields of the header are designated as follows:

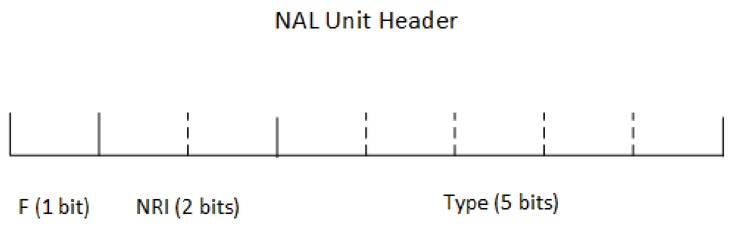


Figure 2: NAL Unit Header Structure

- F: forbidden bit; should always be 0
- NRI: used to indicate if the content of this NAL unit should be used to reconstruct reference pictures in inter picture prediction
- Type: Specifies the NAL unit payload type

Important types of NAL units are different parameter sets that contain relevant information about a given video stream or frame. These can be broken down into sequence parameter sets and picture parameter sets. The sequence parameter set can be transmitted well in advance of the actual video stream and will allow robust protection against the loss of information that change infrequently during a specific session. Picture parameter sets will contain relevant information that will remain unchanged for a particular coded picture. Following the NAL unit will be the VCL coded frames which the decoder will handle based on the type of NAL unit. The H.264/AVC specification also defines a set of profiles and levels which specify different sets of required functional support for decoders. They are designed to support a high degree of interoperability between various different applications of the standard. According to [12], "A profile defines a set of coding tools or algorithms that can be used in generating a conforming bit-stream, whereas a level places constraints on certain key parameters of the bitstream." There exist a number of profiles which provide various features and varying amounts of flexibility and customization points. Some examples of profiles and their applications are:

- Constrained Baseline Profile: low cost applications such as video conferencing in mobile
- Baseline Profile: used in video conferencing
- Main Profile: used in various mainstream broadcast and storage applications
- Extended Profile: intended to be used as a streaming profile
- High Profile: broadcast and disk storage applications (HDTV, Blu-ray, etc.)

For video conferencing applications or streaming from mobile devices such as the system proposed in this paper the constrained baseline or baseline profile are appropriate choices. More detailed information on profile types and their constraints can be found in section A.2 of [3].

In this study, video frames are encoded using the constrained baseline profile. Thus, we have the most basic set of features and can only encode I and P frames (there is no B frame support for this profile) [3]. The type of data being transmitted can be determined based on the NAL unit header. Figure 33 shows how each packet is formatted in the system presented here. Every packet is padded with three bytes of 0x00. This is immediately followed

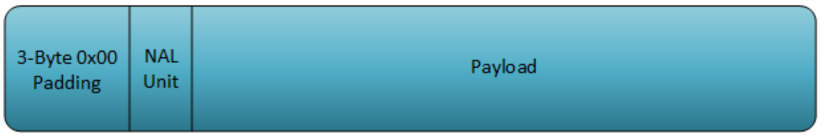


Figure 3: H.264 Packet Format

by the NAL unit for that frame. Finally, we have the encoded data. In the NAL unit, the very first byte is the NAL unit header. In this study, three types of NAL units are defined based on the NAL unit header:

- 0x67: This packet contains a sequence parameter set for the next segment of video
- 0x61: This packet contains the next I-Frame
- 0x41: This packet contains a P-Frame

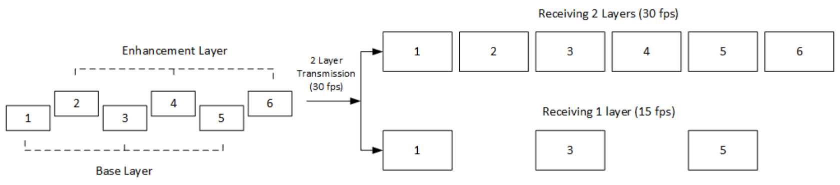
Using NAL header, we determine when a new segment of video is started and weigh the relative importance of each incoming packet.

In the next section, we will discuss the scalable video coding extension of H.264 and compare it with the method of encoding parameter adaptation that we are using to achieve video bandwidth control.

3 Scalable Video Coding vs. Single Layer Video Streams

A major concern in video streaming systems is that of uncertain channel conditions affecting the quality of the video stream. For example, when transmitting a stream at a certain quality having bandwidth B over a congested channel where the bandwidth fluctuates to less than B, the receiving terminal will experience significant degradation of video quality. In order to combat this, scalable codecs have been developed which can alter a certain resolution of the video to fit the channel. This alteration is made to the temporal or spatial resolution of the video or the quality. In H.264/SVC, this is accomplished by removing certain parts of the bit stream in such a way that the underlying streams still represent a valid bit stream for the decoder [2]. For example, a transmitter may send one base layer bit stream and multiple enhancement layer bit streams with the receiver selecting which of these to send to the decoder. In this, video bandwidth can be controlled by choosing only the necessary bit streams to stay within the channel bandwidth. Contrast this with a single layer video stream in which a single bit stream is transmitted which represents a valid stream for a given decoder. In order to have control over the bandwidth of the video, the transmitter must encode the source video with different encoding parameters and the receiving decoder must be able to adapt to these changes. We will compare these two methods of adapting a video stream to varying channel conditions; first, the Scalable Video Coding extension of H.264 as described in [2], and second, using a single layer video stream and altering encoding parameters at the source.

The Scalable Video Coding extension of H.264/AVC inherits all of the base functionality of H.264 with only the necessary added features to achieve scalable video streaming. In this, it supports the primary scalable parameters, being temporal, spatial, and quality resolution. To achieve temporal scalability, the transmitter may send multiple temporal streams divided into a temporal base layer and one or more temporal enhancement layers [2]. One may label these streams as T_0 through T_k. A receiving decoder then simply needs to know which of these access units are valid or invalid for the current stream, starting from 0 through n where $n \leq k$. The ability to partition a stream as such and play only the valid streams is already present in the H.264/AVC standard with the employment of reference picture memory control [2]. The partitioning of a stream into multiple temporal streams is illustrated in Figure 4. In order to achieve spatial scalability, SVC uses multi-layer coding with inter-layer prediction [2]. Multiple layers are trans-



Enhancement Layer
Base Layer
2 Layer Transmission (30 fps)
Receiving 2 Layers (30 fps)
Receiving 1 layer (15 fps)
1
2
3
4
5
6

Figure 4: Temporal Scalability with H.264/SVC [2]

mitted, each corresponding to a specific spatial resolution and referred to by an integer valued dependency identifier between 0 and $d-1$where d is the number of spatial layers [2]. Quality scalability works on the same principle as spatial scalability with the layers transmitted being of the same spatial resolution.

Another method to control the bandwidth of a video stream is to use single-layer coding and manipulate the encoding parameters at the source. In a single-layer coded video stream, one bit stream is encoded and sent to the receiver. This bit stream is of a fixed spatial, temporal, and quality resolution for the entire sequence of video. There is no mechanism built in to the codec to change spatial, temporal, or quality resolution midstream. In this, one can control the bandwidth of the video by using different encoding parameters at the source for different segments of video. We can partition a video stream into multiple segments (sequences) labeled $T_k, k = 0 \ldots n-1$ where n is the number of segments for the given session and T_k is the time instance when the segment k begins. Such a partitioning may be known in advance or can be determined as a function of the channel bandwidth if channel bandwidth can be measured with some reasonable degree of accuracy (for example, using a method like DIChirp as laid out in [8]). At each time instance T_k, the transmitter resets the encoding parameters in such a way that the bandwidth of the video is altered to fit to the channel. When this occurs, a new sequence parameter set is introduced into the stream to signal the changes to the encoded video to the decoder. A sample stream is depicted in Figure 5. The changes to the encoding parameters will insert a delay into the stream for the time it takes to restart the encoding process. To compensate for this and to handle the alteration events at times T_k, Figure 6 shows our proposed receiver architecture. We employed a preprocessor which inspects the NAL unit headers of each incoming packet and determines when the video stream has changed. In this implementation, a new stream arrives when the header is 0x67. At this event, the idle decoder thread will be invoked and set up to decode the next sequence of video. The previously active decoder thread will empty its queue prior to the start of the new

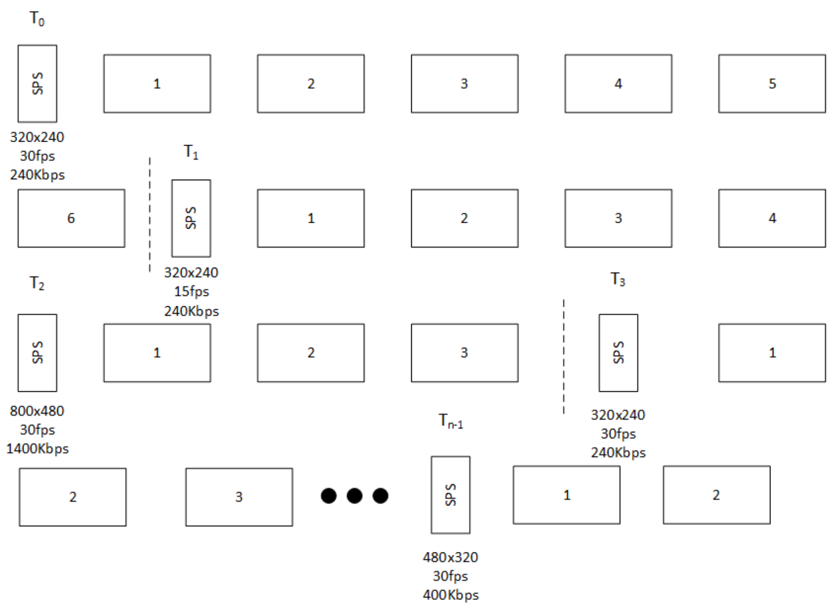


Figure 5: Single Layer Encoder Parameter Switching

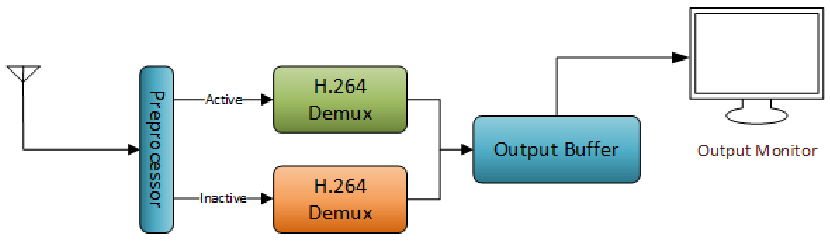


Figure 6: Receiver Architecture

segment of video. This thread can then signal completion and the new thread will take over.

In our system, we decided to use a single layer video stream with the architecture proposed in Figure 6 to achieve video bandwidth control. We are able to accomplish this by using hardware based video codecs; in particular, a hardware H.264/AVC encoder. As a result, we are able to initialize the encoder in real time and there is no CPU overhead when it comes to encoding frames (vs. a software encoder). In addition, this real-time efficiency allows us to achieve rapid encoding parameter switches resulting in minimal delay between video segments. We can control the temporal, spatial, and quality resolution of each video segment, effectively allowing us to control the bandwidth of the video with a range of effects of the output.

4 System Architecture

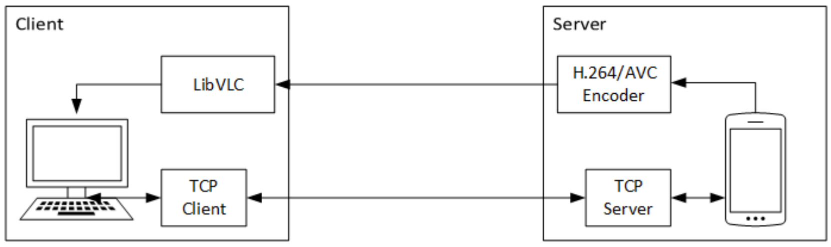


Figure 7: System Architecture

A basic outline of our system architecture is depicted in Figure 7, with the streaming clients, streaming server, and constituent components. The camera system that we have developed consists of a video client and a streaming server. The client connects to the server to request a new streaming session, displaying the video in a media player based on the VideoLAN VLC player. The streaming server is an Android application implemented on Qualcomm MSM8960 hardware. Video is encoded on the device using a hardware H.264/AVC encoder and streamed to the client in raw UDP packets. We decided to deliver video without a higher level wrapper protocol (such as the Real-time Transfer Protocol (RTP) [15] or the Real-Time Streaming Protocol (RTSP) [9]). The server is designed to send a unicast stream to the client connected to it, and a client can view any number of video streams from different servers. The client and server design is detailed in the following sections.

4.1 Client Design

The client is designed as a desktop application consisting primarily of a control center and one or more video windows. The video windows contain a media player for displaying the video, as well as necessary controls for the user to manipulate any of the encoding parameters of the stream.

The media player uses LibVLC, a library used in the popular VLC media player developed by the VideoLAN group. In the media player, we encapsulate the main functionality for configuring the player and playing a certain stream. The media player is configured to be consistent with the architecture described in section 2.2 of this paper. Through configuration of the media player back end, we set up two decoder threads that are used to demultiplex a video stream. This allows us to dynamically change the encoding parameters midstream without seeing a significant effect on video playback. We can then control the bandwidth of the video while still maintaining an optimal user experience. In order to play the video itself, we point the media player to the MRL '`udp/h264://@:[port_num]`'. This MRL specifies a raw UDP stream that should be demuxed using H.264/AVC. The value of "`port_num`" is configured to the port that video is being streamed to.

On top of the media player is a TCP client that provides interaction with the server. When a new video stream is requested, the client attempts to connect to the server, and upon successful connection, is starts the media player. This client then sends all requests to the server and reacts appropriately to responses.

The client is responsible for determining the correct choice of encoding parameters and for managing the bandwidth of the video based on the bandwidth of channel. In this, the client is equipped with the necessary tools to determine the current channel bandwidth and respond to changes appropriately. These decisions should be based on the user's preferences (if an accurate model of the users preferences has been developed) or resort to a default decision function (when learning the users preferences). The client should also be smart enough not to interfere with the user when they make their own decisions about how to scale the video. The method by which we develop user profiles and utilize them will be discussed in section 6 of this study.

4.2 Server Design

Our camera server application runs on a DragonBoard APQ8060A development board utilizing a Qualcomm APQ8060A processor. The application captures live video from one front-facing 8MP camera, at varying spatial and temporal resolutions. When the application launches, we set these to a default of 320x240 at 30 frames per second (fps). Running as a daemon in the application, a TCP server handles all incoming connections from clients and services any requests. On each connection request, a new thread is forked that acts as an interface between the client and server. A handle to the encoder is given to each of these threads to allow them control over the resolutions of the video streams. The handle is encapsulated in an object we call the "encoder activation interface". This object, as the name implies, acts as an interface to the encoder (as well as the camera). Via this interface a consumer may initialize, destroy, and alter an encoder for a certain video stream. This allows the client full control over the parameters of the video and enables it to control the video bandwidth. The server remains agnostic of channel conditions and acts as a slave to the connected clients, reconfiguring the stream as necessary for each request. The reasoning behind this is that the client application learns the users preferences and therefore makes more intelligent decisions about the encoding parameters than the server.

4.3 Session Management

The system contains a communication layer using TCP for messaging between the client and server. This communication layer acts as a session manager. TCP is used for reliable communication of messages between terminals as well as to signal the beginning and end of a streaming session. A streaming session begins once the server accepts a client's connection, and ends when one of the terminals has disconnected. When the client wishes to receive a particular stream from a particular server, it first attempts to make a connection with that server via its TCP client. Upon connection, the server initializes a new thread to service the clients requests. The server thread first starts the encoder, which begins streaming packets containing the encoded H.264 frames. This thread then enters a loop in which it responds to the client's requests until it detects that the client has disconnected. We have defined a very simple protocol for submitting such requests in which the client either requests to alter encoding parameters or stop the video stream. The request to update encoding parameters also serves to start a stream again if it has been previously stopped. To update the encoder, the client sends the following message:

```
<request action="start">
        <width>#</width>
        <height>#</height>
        <fps>#</fps>
        <rate>#</rate>
</request>
```

where "widt" is the new desired width, "height" is the new desired height, "fps" is the new desired frame rate, and "rate" is the new desired video bitrate. To stop the encoder, the request is as follows:

```
<request action="stop" />
```

Upon disconnecting, the thread processing the client's requests will shut down the encoder, stop the video stream, and exit. When a camera update is successful, the server will send a response indicating success. The client will also be notified of any malformations in the request string.

5 Snapdragon Video Framework

The DragonBoard APQ8060A is equipped with numerous hardware based codecs for both audio and video. We decided to use the hardware encoder in order to encode frames in real time. In addition, the extra speed we acquire greatly assists in our video scaling method. We will now describe how the camera server works to capture video frames, encode them, and send them as raw UDP packets.

To access the hardware components, Android provides a wrapper to the OpenMAX Integration Layer called IOMX which can interact with and utilize hardware media codecs. The OpenMAX Integration Layer is a component based API designed to provide a layer of abstraction on top of multimedia hardware and software architecture. It is also designed to give media components portability across a range of devices. Figure 8 illustrates the various layers designed in OpenMAX.

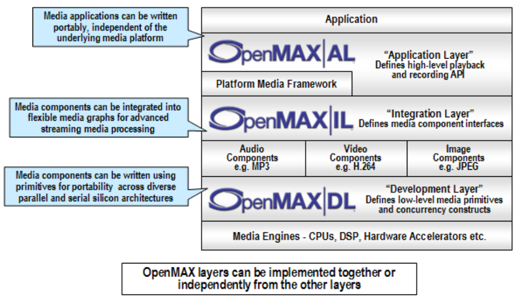

Media applications can be written portably, independent of the underlying media platform
Media components can be integrated into flexible media graphs for advanced streaming media processing
Media components can be written using primitives for portability across diverse parallel and serial silicon architectures
Application
OpenMAX|AL
"Application Layer" Defines high-level playback and recording API
Platform Media Framework
OpenMAX|IL
"Integration Layer" Defines media component interfaces
Audio Components e.g. MP3
Video Components e.g. H.264
Image Components e.g. JPEG
OpenMAX|DL
"Development Layer" Defines low-level media primitives and concurrency constructs
Media Engines - CPUs, DSP, Hardware Accelerators etc.
OpenMAX layers can be implemented together or independently from the other layers

Figure 8: OpenMAX Layers

With the introduction of the stagefright media framework (Figure 9), Google added the OpenMAX IL functionality to the Android operating system, allowing OEMs the ability to provide software hooks that serve as an interface to the hardware for developers.

In addition, Qualcomm has developed and provided to users a sample API that utilizes IOMX to encode and decode various audio and video formats. Our server is leveraging this API to interact with the hardware H.264/AVC encoder present on the device. Qualcomm's implementation can be broken down into a few different levels, as depicted in Figure 10. The lowest levels are the hardware, OpenMAX IL, and IOMX. The API consists of C++ classes which wrap around IOMX, as well as a public interface written in C, providing ease of use for higher level code. In this, one can query available codecs, activate and initialize a session, encode/decode frames, and perform cleanup when a session ends. In our study, the Qualcomm API is used in

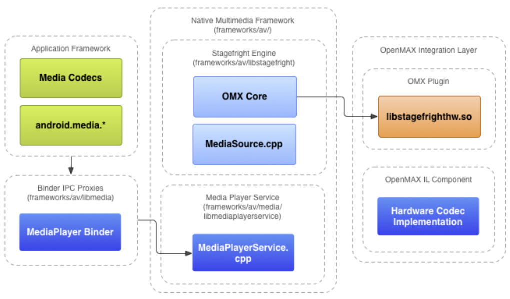

Application Framework
Media Codecs
android.media.*
Binder IPC Proxies (frameworks/av/libmedia)
MediaPlayer Binder
Native Multimedia Framework (frameworks/av/)
Stagefright Engine (frameworks/av/libstagefright)
OMX Core
MediaSource.cpp
Media Player Service (frameworks/av/media/libmediaplayerservice)
MediaPlayerService.cpp
OpenMAX Integration Layer
OMX Plugin
libstagefrighthw.so
OpenMAX IL Component
Hardware Codec Implementation

Figure 9: Stagefright Media Framework

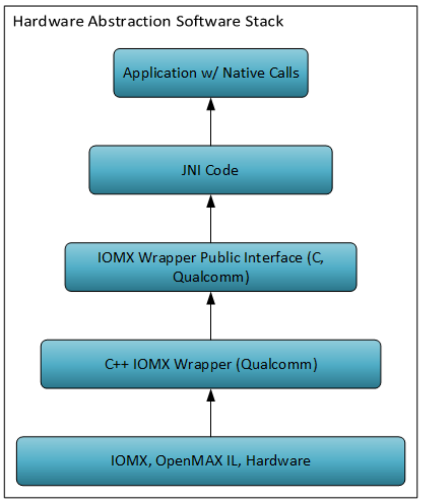

Hardware Abstraction Software Stack
Application w/ Native Calls
JNI Code
IOMX Wrapper Public Interface (C, Qualcomm)
C++ IOMX Wrapper (Qualcomm)
IOMX, OpenMAX IL, Hardware

Figure 10: Encoder Hardware Abstraction

an Android application via the Java Native Interface (JNI) which enables interaction with the public interface of the API. The application developed in this study utilizes this API to quickly and efficiently encode video in real time. In addition, the activation and tear down of an encoding session occurs with very minimal delay.

6 User Profiles and Bandwidth Optimization

The development of user profiles allows the client to make intelligent scaling decisions in line with how the user would have changed the video. This relieves the burden on the user to figure out how video should be scaled in different contexts, making it critical for a widely acceptable system. The learning and prediction mechanism proposed in this study offers a simple, yet effective way to provide dynamically adaptive video streams tailored to each individual.

6.1 Preferences and Profiles

We develop a profile of the user which will determine how video will be scaled when it is no longer optimized to the transmission channel. In this, we have designated the user to be in one of four discrete classes which represent their preferences in relation to video quality:

- Class 0: User prefers high temporal and spatial resolution, no quality preference.
- Class 1: User prefers high temporal resolution; optimize spatial resolution for quality.
- Class 2: User prefers high spatial resolution; optimize temporal resolution for quality.
- Class 3: User prefers optimal quality; configure temporal and spatial resolution appropriately.

Knowing these preferences, we optimize the video bandwidth by weighing the video resolutions with higher or lower priority. This translates into the client treating lower priority parameters more harshly when determining the new encoding parameters.

Transforming these classes to their equivalent binary form we can represent them with a 2 bit value in which the first bit conveys the temporal resolution preference and the second bit conveys the spatial resolution preference. A set bit indicates the desire for a high weight given to this resolution at the expense of quality, and an unset bit indicates the desire to optimize quality resolution at the expense of the resolution in question. In this, we can find the best way to alter the spatial and temporal resolution such that the video bandwidth will fit the channel and align with the user's desires in terms of quality. The creation of these user profiles is accomplished using machine learning; in particular, by solving the classification problem using support vector machines.

6.2 Machine Learning

In order to create a profile for each user we employed supervised machine learning to create a decision function based on the users behavior to predict the class a user falls into. The decision function is calculated using the

LibSVM implementation of a C support vector machine with a radial basis kernel [1].

In supervised machine learning, a training set composed of a series of training samples is presented as input to the learning algorithm, the output of which is a set of coefficients for a decision function. In classification problems such as ours, the training samples are the values of a set of contextual features which we find relevant to the output, and a label for each of these vectors which denotes what class applies at the instant of the sample. The features used in this study are the channel bandwidth and the content type of the video (i.e. high quality medical, talking head, sporting event, etc.). For the two preferences being learned we use two C support vector machines with radial basis kernels. The C support vector classification implementation we are using is defined in [1]. In this, our training set can be defined as a feature vector $x_i \in \mathbf{R^n}, i = 1 \dots l$ and the class label vector $y_i \in \{1, -1\}$ where 1 and -1 indicate the two distinct classes. [1] then solves the primal optimization problem in equation 1.

$$\begin{aligned} \underset{w,b,\xi}{\text{minimize}} \quad & \frac{1}{2}w^T w + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, \dots, l \end{aligned} \tag{1}$$

where $\phi(x_i)$ maps x_i into higher dimensional space and $C > 0$ is a configurable parameter [1]. In addition, [1] solves the dual problem presented in equation 2.

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ \text{subject to} \quad & y^T\alpha = 0 \\ & 0 \leq \alpha_i \leq C, i = 1, \dots, l \end{aligned} \tag{2}$$

where e is a column vector of all ones, $Q_(i, j) = y_i y_j K(x_i, x_j)$ and $K(x_i, x_j) \equiv \phi(x_i)^T\phi(x_j)$ is the kernel function (in our case, a radial basis kernel) [1]. In this, the optimal w satisfies equation 3, and the decision function is defined by equation 4.

$$w = \sum_{i=1}^{l} y_i\alpha_i\phi(x_i) \tag{3}$$

$$sgn(w^T\phi(x) + b) = sgn(\sum_{i=1}^{l} y_i\alpha_i K(x_i, x_j) + b) \tag{4}$$

If we consider the graphical representation of x_i and y_i, we have a multidimensional input space and class labels associated with each vector from x. The support vector machine algorithm attempts to find a hyperplane that separates the two classes with the widest margin, using kernels to make the separation nonlinear. From this hyperplane we get a decision function that is used to predict class labels for new input vectors. In this study, we train two support vector machines to predict the users preferences using a combination of implicit and explicit feedback.

When a new user begins interacting with the system, they are initiated in "learning mode." In learning mode, a change in channel bandwidth will result in a knee-jerk reaction by the system to simply alter the quality resolution of the video, leaving spatial and temporal resolution unchanged. Explicit feedback is obtained from the user by the system asking if the video quality is acceptable. If they answer yes, we add a training sample to our training set. If the user answers no, we take no action. If the user makes a decision to change the video resolutions, we again issue the prompt. This process continues until the training set is sufficiently large enough to accurately train the support vector machines. 3-fold cross validation is performed to ensure accurate selection of the C parameter for the SVM and gamma parameter for the kernel [1]. From this, a decision function is created which we use to make predictions. This process is shown in Figure 11.

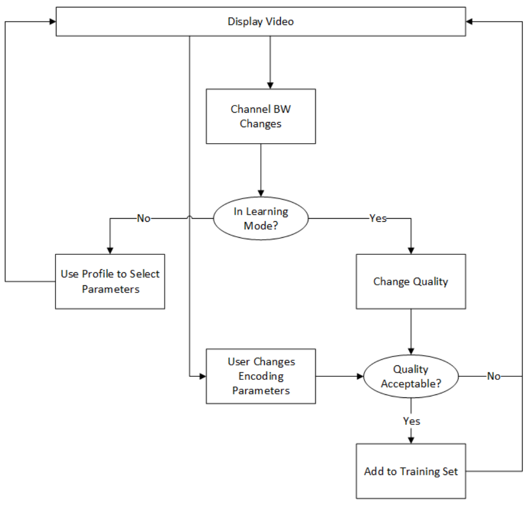


Figure 11: Feedback Process

6.3 Video Bandwidth Determinations

The catalyst for video bandwidth recalculation is when the channel conditions have changed significantly enough to warrant scaling the video. There are numerous methods to estimate the bandwidth of a channel within a certain degree of accuracy, such as DIChirp [8]. We will now lay out how changing the various video resolutions will affect the video bandwidth in our system.

In order to alter the quality resolution we change the compression bitrate of the video. We determine two possible rates, the minimum of which is selected as the final bitrate of the video: maximum rate and optimal rate. The maximum rate can simply be calculated as a function of the known channel bandwidth and an optimization constant:

$$Bitrate_{max} = Bandwidth_{channel} * K \tag{5}$$

where $Bitrate_{max}$ is the maximum allowable bitrate, $Bandwidth_{channel}$ is the bandwidth of the channel, and K is the optimization constant determining the percentage of available bandwidth that is acceptable to fill. Due to limitations in our encoder, the max bitrate must be calculated as a function of the frame rate. We are using the following calculation:

$$Bitrate_{max} = \frac{Bandwidth_{channel}}{\frac{T}{15}} * K \tag{6}$$

where T is the frame rate of the video. The optimal bitrate is found as a function of the spatial resolution and an optimization constant:

$$Bitrate_{opt} = width * height * Q_{max} \tag{7}$$

where $Bitrate_{opt}$ is the optimum bitrate for the given spatial resolution, $width$ and $height$ are the dimensions of the video, and Q_{max} is the optimization constant which can be configured to find the highest necessary bitrate to deliver high quality video at a given spatial resolution. The best choice of bitrate is either the maximum allowable or maximum necessary bitrate for the given channel, spatial resolution, and temporal resolution, which is determined as the minimum of these two bitrates:

$$Bitrate_{out} = \min(Bitrate_{max}, Bitrate_{opt}) \tag{8}$$

This equation gives us the highest achievable bitrate for delivering the best possible quality to the client.

Determining the temporal and spatial resolution will affect the quality resolution such that higher values will result in lower quality once the bitrate has reached $Bitrate_{max}$. We inspect the users class in order to set these two parameters, as the class tells us if the temporal/spatial resolution should be optimized or if the quality should be optimized. For temporal resolution, we limit the possible values to 30 and 15. In the case where the temporal resolution bit is set in the users class, we simply set the frame rate to 30fps, and if unset, change the frame rate to 15 fps. If the spatial resolution bit is set, the spatial resolution is changed to the maximum value. When the bit is unset, we first calculate $Bitrate_{max}$, then reduce the spatial resolution until:

$$Bitrate_{opt} \leq Bitrate_{max} \tag{9}$$

This will result in a spatial resolution that will be small enough to provide optimal quality video.

6.4 Scaling Decisions

The way that video bandwidth is optimized depends on the class that the user falls into for the given feature set at the time T_k (the moment when the encoding parameters of the k^{th} segment of video are selected). When channel bandwidth changes significantly enough, the first action we take is to predict the users preferences using the two decision functions we have already generated. This provides us with a bitmask that serves as the users class; the primary purpose of this class is to specify the order in which to scale the encoding parameters. This ordering also depends on whether or not the video bandwidth is increasing or decreasing. The chart given in Figure 12 depicts the possible decisions that can be made about the video bandwidth. We can then successfully adapt to channel bandwidth changes and alter the encoding parameters in such a way that the users desires are fulfilled. In the next sections we will present our experimental set up and results of our experiments.

7 Experimental Results

To validate the cognitive video scaling solution, we have developed a test bed as shown in Figure 13. We created a point to point connection between the client and server by setting up the client machine as a DHCP server and connecting it directly to the embedded video server, allocating it an IP address on an arbitrary subnet. We are using dummynet , a widely used network emulator, to emulate the behavior of internet in the lab environment. With dummynet, one can control the traffic over a specific channel by limiting bandwidth, inserting packet losses, inserting delay, etc. In order to simulate

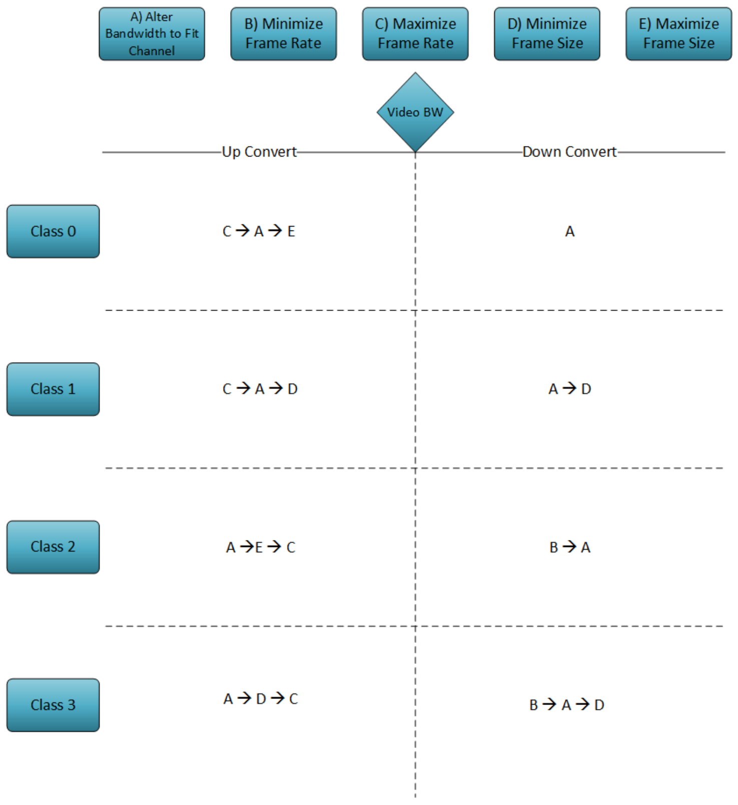


Figure 12: Bandwidth Decisions

bandwidth change detection the client simply reads from a file that contains channel bandwidth information. In our test set up we developed a test application which simultaneously sets the bandwidth of the channel to varying values at certain intervals using dummynet, and writes this bandwidth to a flat file that the client can read from. With this, we are able to implement some of the conditions of a real network and be aware of the bandwidth in the client application.

Our first experiment tests the ability of the system to control the bandwidth of the video by altering the spatial resolution. Channel bandwidth is kept constant, as well as frame rate which is kept at 30 frames per second. The spatial resolution is changed from 800x480 to 480x320 to 320x240. We used Wireshark to capture and display the bandwidth of the video, obtaining results in Figure 14. As the spatial resolution is changed, the bandwidth of the video changes accordingly. One can easily deduce that this change is directly proportional to the change in resolution. For example, when the spatial resolution changes from 800x480 to 480x320, the total pixel ratio and

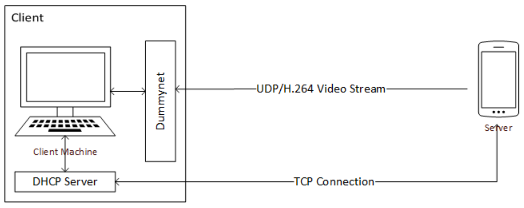


Figure 13: Camera System Test Bed

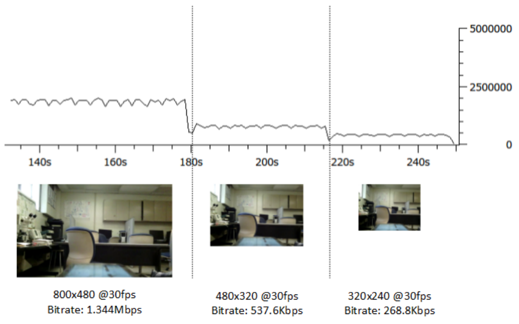


Figure 14: Spatial Resolution Experiment

the ratio of video bandwidth are equivalent:

$$\frac{480 * 320}{800 * 480} = \frac{76800}{384000} = 0.4$$

$$\frac{537.6Kbps}{1344Kbps} = 0.4$$

These results indicate that we have successfully demonstrated control over the video bandwidth by altering the spatial resolution of the video.

In our next test, we showed that we can control video bitrate by altering the amount of compression (number of bits per pixel), resulting in a change in video quality. We tested the systems response to changes in channel bandwidth by altering the bandwidth from 1600Kbps to 800Kbps to 400Kbps. Spatial resolution was kept constant at 800x400 and temporal resolution was kept constant at 30 fps. The resulting changes in video bitrate, as well as playback quality, are depicted in Figure 15. As can be seen from Figure 15, the system successfully and immediately responds to changes in channel bandwidth by reducing the quality resolution of the video. In addition, the reduction in video bitrate is directly proportional to the reduction in available bandwidth.

Finally, we tested the systems ability to determine the users preferences and scale the video appropriately when channel bandwidth changes. For the

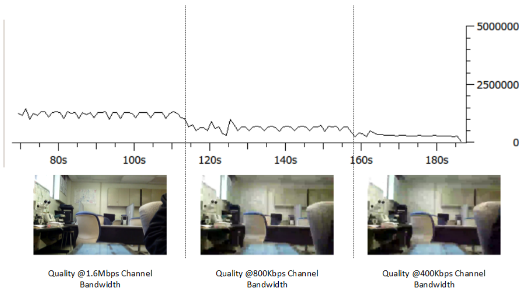


Figure 15: Quality Resolution Experiment

purposes of this study, we provided the learning mechanism with an arbitrary training set with the intent to prove the systems ability to properly learn and make predictions. By using a predefined training set we know in advance what classes should be predicted, allowing us to validate the accuracy of the cognitive mechanism by comparing the experimental predicted values with the expected values. The training set used is given in Table 1. The class distribution for this training set is depicted in Figure 16. The "Content

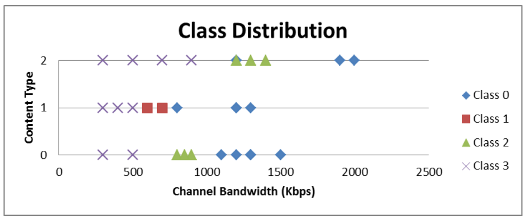


Figure 16: Class Distribution

Type" feature is used to classify different scenes that are transmitted in real world scenarios. For example, a content type of 0 may be a video conferencing application with a "talking head" based scene, while a content type of 2 may be a medical based scene that requires extremely high bandwidth and video quality. In this experiment, we used 3 different content types, giving us the approximated expected prediction values in Table 2. We trained the support vector machines and tested the learning mechanism by setting the bandwidth to 1600Kbps, 800Kbps, and 400Kbps while viewing video streams with content types 0, 1, and 2. For equation 5 we selected the K parameter to be 0.85 and for equation 7 we selected the Q_{max} parameter to be 3.5. The resulting bandwidth changes, encoding parameter changes, and class predictions are given in Figures 17-19. In all cases, the video bandwidth was adapted when the channel bandwidth changed. In addition, the video bandwidth was consistently kept at 85% of the channel bandwidth (as a result

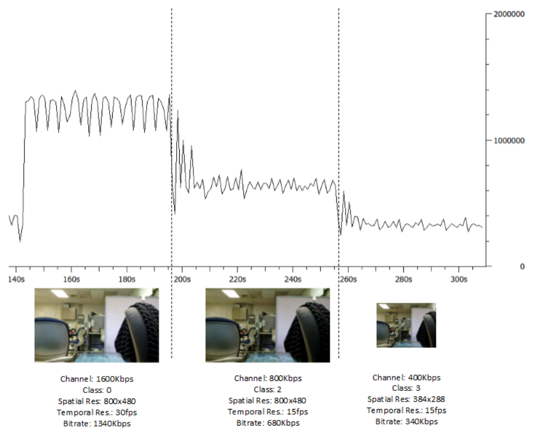


Figure 17: Predictions With Content Type 0

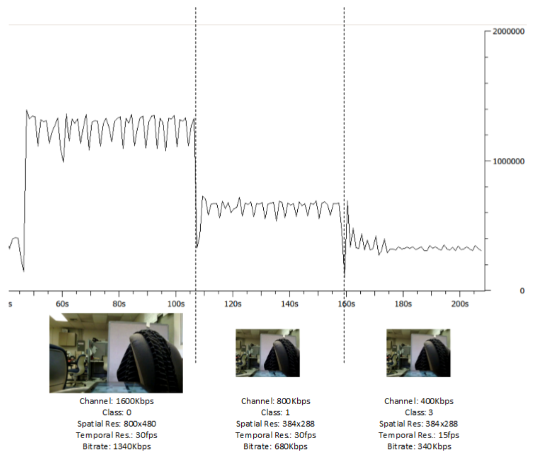


Figure 18: Predictions With Content Type 1

of our selection of K). In order to validate the accuracy of the predictions, Table 3 compares the expected class values and the predicted class values at time instances T_0, T_1, and T_2, where T_0 is the instant when the channel bandwidth changes to 1600Kbps. As the table indicates, the support vector machines predicted the correct class with 100% accuracy. In addition, the resulting changes to the encoding parameters followed the changes defined in Figure 12. When class 0 was predicted, the spatial and temporal resolutions were kept high at the cost of fewer bits per pixel. When class 1 was predicted, the temporal resolution was kept high and the spatial resolution was reduced. The prediction of class 2 resulted in a loss in temporal resolution with the spatial resolution being kept high. Finally, when class 3 was predicted, the spatial and temporal resolutions were reduced, resulting in greater quality

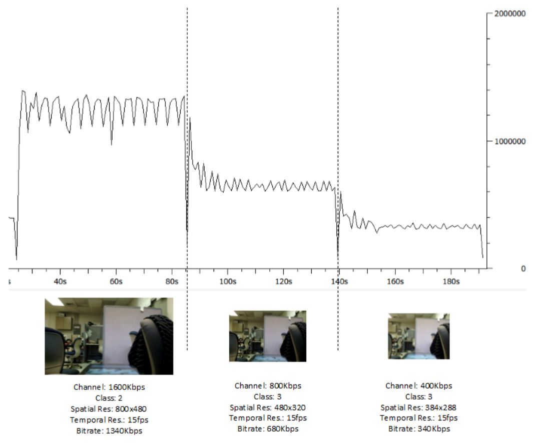


Figure 19: Predictions With Content Type 2

with more bits per pixel.

8 Conclusion

As the volume of internet traffic related to video streaming increases, the importance of having exceptional bitrate adaptation schemes grows. In addition, these schemes should be aware of not only the channel bandwidth, but the surrounding context of the video being streamed. This context encapsulates the content type of the video, and can extend into other dimensions such as amount of motion, geospatial location, and more. We have presented a solution that takes rate adaptation beyond simply changing the quality of the video when the channel bandwidth becomes limited. Our system determines the users preferences about video quality, taking into account if the user prefers a drop in temporal or spatial resolution versus quality resolution. We have demonstrated the ability to adapt to channel bandwidth changes by altering these resolutions. In addition, we have shown that by using support vector machines, we can learn the users preferences and successfully adapt the video bandwidth in line with these preferences. Such a system is a viable solution to the changing atmosphere of video providers, contexts, and the increasing and diverse consumer base.

Acknowledgements

This study in part has been supported by Qualcomm University program.

References

[1] C. Chang and C. Lin. Libsvm: A library for support vector machines.

[2] D. Marpe H. Schwarz and T. Wiegand. Overview of the scalable video coding extension of the h.264/avc standard. *IEEE Transactions on Circuits and Systems for Video Technology*, September 2007.

[3] ITU-T Recommendation H.264. Advanced video coding for generic audiovisual services. May 2003.

[4] I Urteaga et. al. I. Unanue. A tutorial on h.264/svc scalable video coding and its tradeoff between quality, coding, efficiency, and performance.

[5] S. Mascolo L. De Cicco and V. Palmisano. Feedback control for adaptive live video streaming. *MMSyss 11 Proceedings of the Second Annual ACM Conference on Multimedia Systems*, 2011.

[6] J. Ohm. Advances in scalable video coding. *Proceedings of the IEEE*, 93, January 2005.

[7] O. Oyman and S. Singh. Quality of experience for http adaptive streaming services. *IEEE Communications Magazine*, April 2012.

[8] Y. Ozturk. Dichirp.

[9] H. Schulzrinne. Rfc 2326 real time streaming protocol (rtsp). April 1998.

[10] T. Einarsson et. al T. Lohmar. Dynamic adaptive http streaming of live content. 2011.

[11] M. Hannuksela T. Stockhammer and T. Wiegand. H.264/avc in wireless environments. *IEEE Transactions on Circuits and Systems for Video Technology*, July 2003.

[12] G. Bjontegaard T. Wiegand, G. J. Sullivan and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, July 2003.

[13] Y. Guo et. al V. Adhikari. Unreeling netflix: Understanding and improving multi-cdn movie delivery.

[14] S. Wenger. H.264/avc over ip. *IEEE Transactions on Circuits and Systems for Video Technology*, 13, July 2003.

[15] S. Wenger M.M. Hannuksela T. Stockhammer M. Westerlund and D. Singer. Rtp payload format for h.264 video. *Network Working Group*, February 2005.

[16] Apple whitepaper. Http live streaming overview.

[17] A. Zambelli. Iis smooth streaming technical overview. *Microsoft whitepaper*, March 2009.

Table 1: Training Set

Channel Bandwidth (Kbps)	**Content Type**	**Class Label**
1200	0	0
1300	0	0
800	0	2
500	0	3
900	0	2
300	0	3
1100	0	0
1300	0	0
1500	0	0
850	0	2
500	1	3
1200	1	0
600	1	1
1300	1	0
300	1	3
400	1	3
800	1	0
600	1	1
700	1	1
1200	2	0
500	2	3
1200	2	0
2000	2	0
1300	2	2
300	2	3
900	2	3
1400	2	2
1900	2	0
700	2	3
1200	2	2

Table 2: Expected Prediction Values

Channel Bandwidth (Kbps)	**Content Type**	**Expected Class**
0-500	0	3
500-1000	0	2
1000+	0	0
0-500	1	3
500-800	1	0
800+	1	0
0-900	2	3
900-1700	2	2
1700+	2	0

Table 3: Expected vs. Predicted Classes

Segment	**Content Type**	**Expected Class**	**Predicted Class**
T_0	0	0	0
	1	0	0
	2	3	3
T_1	0	2	2
	1	1	1
	2	3	3
T_2	0	3	3
	1	3	3
	2	3	3