Dynamic Video Buffer Compensation

Evan Tan[1] Chun Tung Chou[2]

[1] NICTA and University of New South Wales, Australia
evan.tan@nicta.com.au
[2] University of New South Wales, Australia
ctchou@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-1001
February 2010

Abstract

Decoder buffer underflow is one of the main issue with video streaming. This is mainly brought about by the time varying network conditions. It has an impact on the user perceived video quality as it stops video playback to allow for rebuffering. Having a large enough buffer would, in theory, solve this issue, but it brings about delays that might violate the application's delay requirement. A way to tackle this issue is to do buffer compensation, this is done by allowing more video data to be transmitted and stored in the buffer. What we propose to do is to increase the video frames being sent by increasing the encoding frame rate at the encoder and decrease the frames being played by decreasing the playout frame rate at the decoder to reduce the probability of a decoder underflow. However, adjusting these two parameters would have an adverse effect on both the frame quality and the playout quality. So we further propose an optimization framework based on Lyapunov drift analysis that adjusts the encoding and playout frame rates to maximize the frame and playout qualities as well as to maintain the buffer at an acceptable level. Simulation results shows that our proposed framework obtained significant improvements over a typical setup of fixed encoding and playout frame rates by up to 70% in video utility.

1 Introduction

Streaming video provides a way to both preview and view video content with a minimal startup delay. This enhances the viewing experience for the users as they can watch just enough of the video to decide whether to continue watching or stop. This feature of streaming helps to optimize the network resource allocation and potentially reduces the 'badput' [24] in the network. Furthermore, the client-end does not need to store the entire video for playback. As a result, video streaming not only enables an interactive video capability but is quickly becoming a popular choice for transmitting video content in networks both for the provider and users.

However, streaming video over a network is a challenging task as it needs to handle the time-varying network conditions and maintain the quality of the received video. Note that the video quality here not only refers to the *frame quality* (i.e. the clarity of each video frame) but also *playout quality* of the video (i.e. the continuity of video playback).

A typical way to handle the varying network conditions is to buffer video at the client-end (i.e. the decoder). Video can be buffered by delaying the video playback, this allows for a smoother video quality at the expense of viewing delay. From a network's perspective, it is desirable to have large buffers to deal with the large variations in the network condition. This is because every time the network bandwidth drops or a packet loss occur, the video data in the buffer tends to drop. A buffer underflow might occur with a sufficiently long series of bandwidth drops and packet losses. In theory, a large enough buffer would solve this issue. However, from an application's perspective, larger buffers translate to a larger viewing delay which may violate the delay requirements of the application. Excessive delays will impact on the user perceived quality and the delay requirement varies greatly for different video applicaions. One way video broadcasts tend to have delays from several seconds to tens of seconds, which is fairly large, while real-time video transmission applications like video telephony usually have a delay of sub 400 ms [19].

Each bandwidth drop or packet loss event lowers the buffer level and increases the possibility of a buffer underflow. A way to avoid this is to send more video data whenever a bandwidth drop or loss event occurs to compensate for the drop in buffer level. In this paper, we call this *buffer compensation.* This can be done by changing the encoder parameters so that more video data will be transmitted and stored in the buffer. Using buffer compensation, buffer sizes can be small enough to meet the application delay requirements without sacrificing too much on the video quality.

In this paper, we aim to perform decoder buffer compensation that caters for the dynamically changing network throughput while simultaneously maximizing the quality of the received video. To achieve this, we propose a framework that dynamically compensates the decoder buffer. The main idea behind the framework is that whenever the buffer level drops, the framework will adjust the encoding frame rate (i.e. coded frame size) of the encoder to increase the number of frames being transmitted so that the decoder buffer can be compensated. Simultaneously, at the decoder, the playout frame rate will adjusted to decrease the rate of video data being removed from the decoder buffer.

Increasing the encoding frame rate will mean that more frames will be sent based on a given bit-rate from the transport layer. This implies that the frame sizes would be smaller than before to accommodate the higher frame rate. While, this will help to compensate the drop in buffer level, it results in a lower frame quality. Similarly, reducing the playout frame rate would result in a drop in the playout quality of the video. Thus, the adjustment of the frame rates at both encoder and decoder requires some careful coordination between them.

In order to systematically approach this problem, we formulate an optimisation problem with three (possibly conflicting) objective: maintaining a certain buffer level (i.e. a steady buffer occupancy), maximizing frame quality and maximizing playout quality. To achieve these three objectives simultaneously, the framework uses optimization policies derived from Lyapunov drift analysis [16]. We show that these policies can be decoupled to encoder and decoder optimization policies and that under appropriate conditions, the decoupled problems are convex. That is, the encoder will adjust the encoding frame rate while the decoder independently adjusts its playout frame rate. The only information needed to be passed from the decoder to the encoder is the decoder buffer level and the network loss rate. The derived optimization policies maximizes the video quality, in terms of both the frame and playout quality. It also adapts to the time-varying network throughput while at the same time ensuring a steady decoder buffer occupancy level.

Our contributions are two-fold:

1. We perform decoder buffer compensation by joint adjustment of encoding and playout frame rates.
2. We formulated an optimization problem using Lyapunov drift analysis to adjust the frame rates, thus ensuring a close to optimal video quality while maintaining a steady decoder buffer occupancy. Additionally, under certain assumptions on the network conditions, the Lypunov drift method provides a bound on the performance.

The proposed framework has a number of advantages. The framework also works on top of the transport layer and can function with any transport protocol or any protocol which can provide network congestion information. Furthermore, the dynamic Lyapunov optimization policies ensure that the framework can adapt to any type of network. It also inherently prevents buffer overflows and provides guarantees in performance.

In the next section (section 2), we discuss several related works on video streaming. In section 3, we present a more detailed version of the buffer compensation problem and formulate an initial optimization problem. Section 4 discusses Lyapunov drift analysis and presents the Lyapunov drift optimization framework that is based on the initial optimization problem formulated in section 3. Section 5 then discusses the video utilities to be used in the optimization framework. Section 6 presents some simulation results of the framework and finally section 7 concludes this paper.

2 Related Work

Our proposed framework falls into the category of video rate control. Some examples of typical video rate control techniques are those that have been proposed for the MPEG-4 and H.264/AVC non-normative standard by Lee et al [9] and Li et al [13] respectively. These techniques chooses the most suitable quantization parameter (QP) that would enable a video to meet the bandwidth and encoder buffer constraints, a process known as QP selection. The bandwidth and buffer constraints are modeled using a fluid buffer model, which is calculated based on the available bit-rate and the remaining free encoder buffer space. The resulting rate is then fed into a derived rate-QP model to get a suitable QP for the video.

A newer video rate control technique is in x264, a popular open source H.264/AVC software, by Merritt and Vanam [15]. x264's rate control, like H.264/AVC's non-normative rate control, performs QP selection to meet the bandwidth constraint. However, unlike H.264/AVC's rate control, its algorithm is mainly empirically derived. Moreover, it utilizes newer concepts such as a fast pre-motion estimation of the frames to estimate scene complexity and performing QP compensation for future QPs to account for mispredictions.

Adjusting the encoding frame rate for rate control has been proposed by Song and Kuo [20] for H.263+. The motion complexity for each frame is estimated. The encoding frame rate is then adjusted by some predefined amount if the motion complexity exceeds certain thresholds. Generally, the higher the motion complexity, the lower the encoding frame rate will be. The goal of their approach is to enable better quality video to be transmitted at low bit-rates.

These video rate control techniques all function within the encoder and attempt to produce a video stream at a certain bit-rate. However, they do not directly consider the decoder buffer occupancy.

The next category of video streaming techniques is cross-layer video adaptation. The main goal of most of these techniques is to obtain accurate information about the network so that the encoder can adapt and send video data that can meet the network constraints.

As the transport layer performs congestion control, a number of techniques by Yan et al [23], Zhu et al [25] and us [22] have been proposed that integrates video rate control with a transport layer protocol. These transport integrated techniques usually try to ensure that the output rate of the video is *TCP-friendly*, which means that the video streams will compete fairly with TCP traffic. Given that the majority of the traffic in most networks are TCP traffic, TCP-friendliness is a desirable property for a video stream. Additionally, these techniques also attempt to derive an output video rate that also improves the frame quality of the video.

Zhu et al [25] also targeted decoder buffer occupancy. A virtual buffer management algorithm is established where the network is represented as a virtual buffer with an input rate and output rate (i.e. the network throughput) along with the encoder and decoder buffers. Using this, the encoder calculates the required sending rate that would avoid a decoder buffer underflow. Should the decoder detect its buffer to be close to an underflow, the encoder then temporarily violates TCP-friendliness to send more video data to the decoder to avoid underflow. This TCP-friendliness violation is then compensated for by sending future video data at a lower rate to maintain long-term TCP-friendliness. One assumption that was made is that the virtual buffer is a sufficient representation of the network, this may not be true as extra video data sent to avoid an underflow might not even reach the decoder.

Furthermore, these TCP-friendly techniques attempts to send video data at a higher rate to improve video quality. This possibly exacerbates network congestion which would result in higher packet losses. Thus, the user might end up with a lower received video quality compared to sending it at a lower TCP-friendly rate. This problem is partially due to the loose definition of TCP-friendliness by Floyd et al [6] as "generally within a factor of two of the sending rate of a TCP flow under the same conditions". Some cross-layer video adaptation techniques avoid this issue by either using a bandwidth estimation tool or by defining their own congestion model of the network.

Li et al [10] based their work on a bandwidth estimation tool to gather channel statistics. These channel statistics are then fed into a buffer model which determines the best rate to stream the video at. While their approach results in a more network-friendly video rate, they do not directly try to maximize the video quality. Also, the mean time between buffer underflow is used as a parameter for their buffer model, this is difficult to determine for a given video application.

Zhu et al [26] derived delay probabilities with respect to the video rate based on a M/M/1 queuing model and showed it to be a convex function. Furthermore, the wireless link capacity is calculated based on the signal-to-interference-plus-noise-ratio (SINR) and used as the capacity constraint. A video distortion function is then paired with the delay probabilities and the capacity constraint to form a convex optimization problem which is subsequently solved using conventional convex optimization solvers. While the M/M/1 model gives a nice closed-form solution, its precision is questionable. Whether it is possible to derive a convex delay function with respect to the rate using a more general queuing model has not been explored by the authors yet.

Li et al [11] and Dai and Loguinov [5] used network utility maximization (NUM). NUM solves a global network optimization problem by decomposing it into a smaller per node local optimization problems. Each node then solves its own miniature optimization problem and, through message passing, the global network optimization problem is solved. By solving the global network optimization problem, each node within the network uses a fair share of the bandwidth that maximizes its own utility function. For video, the distortion function replaces the utility and the inverse NUM is solved instead. However, NUM solves a static optimization problem, it does not explicitly take into account of the time varying properties of the network.

Defining a congestion model of the network would result in a more precise estimate of the network congestion. However, these techniques tend to require every node in the network to participate in the congestion signaling, this would require some fundamental architectural changes to the routers. Thus, these techniques tend to be restricted to small networks and would have difficulty scaling up to large heterogeneous networks.

The last category of video streaming techniques that our framework falls into, is the adaptive media playout (AMP). AMP techniques all adjust the playout rate at the decoder to avoid a decoder buffer underflow.

The concept of adjusting the playout rate to stabilize the decoder buffer occupancy was first proposed by Kalman et al [8]. The playout rate of the decoder is adjusted by a predetermined fixed percentage every time the decoder buffer falls below or exceeds a certain level. As the playout rate is adjusted by a constant value, the approach might not adapt to the different loss characteristics of different networks.

Chuang et al [4] adjusts the playout rate to reduce the probability of a buffer underflow. The underflow probability is modeled as an exponential distribution with its parameters calculated based on the arrival and departure rate at the base station and the decoder buffer occupancy. However, their technique is specific to networks with a central base station as it requires certain information from the base station.

The closest to our work is proposed by Li et al [12]. They formulated a packet scheduling framework based on Markov decision process and coupled it with a content-aware adaptive playout. Just before the video is transmitted from the source, video packets which will possibly miss the playout deadline are dropped by the scheduler based on the deadline, the channel condition and the playout rate at the decoder. At the same time, the decoder playout rate is adjusted to ensure a minimum number of dropped video packets.

Given the similarities with our work, we like to highlight three key differences. Firstly, our framework allows for real-time video, while the technique proposed in [12] is too computationally expensive to be computed on-the-fly and targets pre-stored video applications. Secondly, [12] uses a packet scheduler to drop video packets which are already been encoded, while we directly adjust the encoding frame rate at the encoder. This allows the encoder to adapt more readily to the channel conditions and also has an additional effect of not affecting the source transmission rate as we only adjust the number of frames being sent at a specific transmission rate. Lastly, we used Lyapunov drift analysis to formulate our problem, this enables our problem to be decoupled into two efficient sub-problems for encoder and decoder. In contrast, [12] uses a Markov decision process mainly for packet scheduling.

3 Overview

3.1 Video Transmission Scenario

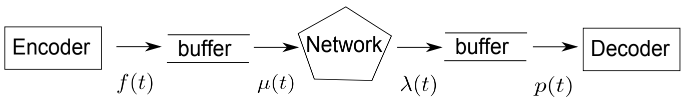


Figure 3.1: Encoding-decoding flow. All rates are in frames per second.

Figure 3.1 shows a typical video transmission scenario. The encoder encodes video at a rate of $f(t)$ frames per second (fps) at time t into the encoder buffer. The network transport protocol then transmits the video data from the encoder buffer into the network at a rate of $\mu(t)$ fps. The transmitted video data will be received at the decoder at a throughput of $\lambda(t)$ fps in its buffer. The decoder then proceeds to playout the received video data from the decoder buffer at a rate of $p(t)$ fps. Video data arriving after the playout deadline are assumed to be lost.

Notice that $\lambda(t) \leq \mu(t)$ due to the delays and losses that may occur while traversing the network (e.g. due to congestion). A side-effect of this is that whenever a sufficiently long series of consecutive network loss or delay occurs or when the bandwidth drops, the decoder's buffer occupancy will drop. This makes the decoder more sensitive to network jitters and further losses, and may cause it to underflow. Thus, there is a need for buffer compensation to handle this issue.

One way to do this is to adjust the encoding frame rate $f(t)$ (i.e. reduce the frame size) as well as the playout frame rate $p(t)$ to compensate for the drop in decoder buffer occupancy. This will help to reduce the probability of buffer underflows and allow for a smoother video playout. Furthermore, adjusting the frame rates will not increase the video bitrate and thus would not create any unnecessary network congestion.

3.2 Design Considerations

For frame rate adjustment to be useful as a buffer compensation technique, it should take the following into consideration:

1. *Frame Quality.* The picture quality of the video. It is well-known that the picture quality of certain types of video are easier to maximize given a rate constraint (e.g. low motion video).
2. *Playout Quality.* The frame rate quality of the video being played to the user. It has been shown by Ou et al [18] that high motion videos need higher frame rates to attain user satisfaction.
3. *Steady Buffer Occupancy.* The process of maintaining a certain level of buffer occupancy regardless of other factors. This is important as it determines how much jitter the decoder can tolerate and provides a continuous playout as it ensures that the buffer does not underflow or overflow.

Ideally, a frame rate adjustment technique would adjust to maximize video and playout quality while maintaining a steady buffer occupancy. However, maintaining a steady buffer occupancy would sometimes mean sacrificing on video and/or playout quality.

3.3 Optimization Problem

Following from our above discussion, we now formulate an optimization problem to capture the design considerations. To simplify the problem, we assume in this paper that $f(t) = \mu(t)$ and $\lambda(t)$ is an random variable that depends on $\mu(t)$. We also assume that there is an throughput function $F(\mu(t))$ that provides the *expected* arrival rate $\mathbb{E}\{\lambda(t)\}$ based on a certain $\mu(t)$, formally:

$$F : \mu(t) \mapsto \mathbb{E}\{\lambda(t)\} \tag{3.1}$$

Since we assumed $f(t) = \mu(t)$, this would mean (3.1) becomes:

$$F : f(t) \mapsto \mathbb{E}\{\lambda(t)\} \tag{3.2}$$

The throughput function $F(f(t))$ is only assumed to be continuous. However, in this paper, we specify,

$$F(f(t)) = (1 - l(t)) \times f(t) \tag{3.3}$$

Where $l(t)$ is the loss probability of the network, this can be calculated based on the history of packet losses and is assumed to be an i.i.d random variable [1] here. We further assume that $\lambda(t)$ is conditionally independent of the past history given the current $f(t)$ and $l(t)$. Let the utility functions $g(f(t))$ and $h(p(t))$ represent the frame and playout qualities respectively. Then, the optimization problem would be:

$$
\begin{aligned}
\text{Maximize:} \quad & w_1(F(f(t)) - p(t)) + w_2 g(f(t)) \\
& + w_3 h(p(t)) && (3.4) \\
\text{Subject to:} \quad & (p(t), f(t)) \in \Lambda && (3.5) \\
& 0 \leq p(t) \leq p_{max} && (3.6) \\
& 0 \leq f(t) \leq f_{max} && (3.7)
\end{aligned}
$$

Where $w_i > 0$, $i \in \{1, 2, 3\}$, are some weighting constants. It can be seen that the three terms in the objective function (3.4) tries to maximize, respectively, the buffer occupancy, video quality and playout quality.

The first term maximizes the buffer occupancy by maximizing the *difference* between the frames received and the playout rate, this helps to minimize the probability of a buffer underflow. It can be seen that the first term can only be maximized by having a high $f(t)$ and low $p(t)$. However, a high $f(t)$ would result in a lower utility from $g(f(t))$ and $h(p(t))$ would also give a lower utility due to a low $p(t)$. Thus, the optimization policy would need to find the proper trade-offs to maximize (3.4).

The constraint (3.5) prevents the buffer overflow. The set Λ contains all the frame rate vectors that the system can support without causing a buffer overflow. Λ is obtained by considering all possible allocations of $p(t)$ and $f(t)$). Note that the exact knowledge of Λ is not needed to calculate the policy, it is only needed to prove the analytical properties of the algorithm.

Ensuring that the buffer does not overflow can also be termed *buffer stability.* More precisely, let $U(t)$ represent the decoder buffer occupancy. We then analytically define buffer stability in this paper as:

$$
\mathbb{E}\{U\} \triangleq \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{U(\tau)\} < \infty \qquad (3.8)
$$

It is difficult to obtain a stable solution using convex optimization tools [3] on the above problem as it considers the stochastic properties of the network through $F(f(t))$. However, the goal in this part is to formulate an optimization problem to capture the design considerations which is shown in (3.4).

To obtain an optimal solution that can dynamically adapt to the stochastic changes and still provide stability would require Lyapunov drift analysis. We will discuss in the next section how we construct a frame rate adjustment technique using Lyapunov drift analysis.

4 Dynamic Lyapunov Drift Analysis

Lyapunov drift analysis is a mathematical tool that is used to provide methods for queue stability in networks. Recent works by Neely et al [7, 16] have ex-

[1] The i.i.d assumption is not always necessary. The derived algorithm will be the same even if it is based on a more general Markov model. See [7] for the derivation technique.

tended Lyapunov drift analysis to provide performance optimization in addition to queue stability. A side effect of this extension is that it gives a *sub-optimal* solution, however, by choosing certain parameters the solution can be made close to the optimal solution. Here we adapt the extended Lyapunov drift analysis into a frame rate adjustment technique.

4.1 Policy Derivation

We first show how we convert the optimization problem presented in section 3.3 into a separate encoder and decoder optimization policies using Lyapunov drift analysis.

We assume that the encoder, decoder and network works in slotted time where each time slot corresponds to the time needed to process a single frame, that is $t \in \{0, 1, 2, \ldots\}$.

The decoder buffer dynamics at each slot will then be:

$$U(t+1) = max(U(t) - p(t), 0) + \lambda(t) \tag{4.1}$$

We next define a Lyapunov function $L(U(t))$ to represent the "energy" of the decoder buffer $U(t)$ at time t. We use the following Lyapunov function:

$$L(U(t)) \triangleq \theta U^2(t) \tag{4.2}$$

Where $\theta > 0$ is a predefined constant. We then define the one-step conditional Lyapunov drift $\Delta(U(t))$ as:

$$\Delta(U(t)) \triangleq \mathbb{E}\{L(U(t+1)) - L(U(t))|U(t)\} \tag{4.3}$$

By squaring the buffer dynamics equation (4.1) and taking expectations of the result, we will get the following expression:

$$\Delta(U(t)) \leq \theta B - 2\theta U(t)\mathbb{E}\{p(t) - \lambda(t)|U(t)\} \tag{4.4}$$

Where $B = f^2_{max} + p^2_{max}$. By subtracting from both sides, the term $V\mathbb{E}\{w_1(\lambda(t) - p(t)) + w_2 g(f(t)) + w_3 h(p(t))|U(t)\}$, which is the expectation of (3.4) scaled by a constant $V > 0$, and by rearranging the terms. We get:

$$\begin{aligned}
\Delta(U(t)) - V\mathbb{E}\{&w_1(\lambda(t) - p(t)) \\
&+ w_2 g(f(t)) + w_3 h(p(t))|U(t)\} \\
\leq \theta B - \mathbb{E}\{&V w_2 g(f(t)) + V w_1 \lambda(t) \\
&- 2\theta U(t)\lambda(t)|U(t)\} \\
- \mathbb{E}\{&V w_3 h(p(t)) + 2\theta U(t) p(t) \\
&- V w_1 p(t)|U(t)\} \\
= \theta B - \mathbb{E}\{&V w_2 g(f(t)) + V w_1 F(f(t)) \\
&- 2\theta U(t) F(f(t))|U(t)\} \\
- \mathbb{E}\{&V w_3 h(p(t)) + 2\theta U(t) p(t) \\
&- V w_1 p(t)|U(t)\}
\end{aligned} \tag{4.5}$$

The key idea of the Lyapunov optimisation is to keep the left-hand-side (LHS) of (4.5) small. Note that (4.5) is the sum of Lyapunov drift $\Delta(U(t))$ and

$-V$ times the expectation of the objective function (3.4). The minimization of Lyapunov drift will ensure that the buffer overflow will not likely occur. The minimization of $-V$ times the expectation of the objective (3.4) will ensure that the objective (3.4) is maximized. Since the minimization of the LHS of (4.5) is hard, Lyapunov optimisation chooses to minimize the upper bound of the LHS of (4.5), which is given by the right-hand-side (RHS) of (4.5).

Also note that in (4.5), the terms are rearranged such that the second term is only a function of $f(t)$ while the last term is a function of $p(t)$ only. Thus, it can be seen from (4.5) that the last term of the right hand side represents the decoder objective function and minimizing this term would provide an ideal playout frame rate. Likewise, the second term of the right hand side represents the encoder objective function and minimizing this term would provide an ideal encoding frame rate. Notice that the frame adjustment policy that is decoupled into separate optimization subproblems for playout frame and encoding frame rate. Furthermore, under appropriate conditions, the decoupled problems are convex. This makes the problem easier and more flexible to solve.

How close to optimality the solution would reached will depend on the parameter V, with the solution reaching close to the optimum as $V \to \infty$. However, a larger V would result in a larger decoder buffer occupancy. For more details on how V affects the solution optimality, see the performance bounds derivation of this policy in Appendix A.

4.2 Frame Adjustment Policy

We will now state the frame adjustment policy based on the derivations from the previous section.

Playout Frame Rate

At each time slot, the decoder will observe the current buffer occupancy $U(t)$ and choose $p(t)$ as the solution of the following optimization:

$$\begin{aligned} \text{Maximize:} \quad & Vw_3 h(p(t)) + 2\theta U(t)p(t) - Vw_1 p(t) \\ \text{Subject to:} \quad & 0 \leq p(t) \leq p_{max} \end{aligned} \tag{4.6}$$

Note that the objective of (4.6) is directly taken from the third term of (4.5). The decoder will then adjust its playout rate to $p(t)$. It can also be seen from (4.6) that the choice of $p(t)$ depends mainly on the playout quality function $h(p(t))$ and the current decoder buffer occupancy $U(t)$.

Encoding Frame Rate

At each time slot, given the decoder buffer occupancy $U(t)$ and the network loss probability, the encoder will choose $f(t)$ as the solution of the following optimization:

$$\begin{aligned} \text{Maximize:} \quad & Vw_2 g(f(t)) + Vw_1 F(f(t)) \\ & - 2\theta U(t)F(f(t)) \\ \text{Subject to:} \quad & 0 \leq f(t) \leq f_{max} \end{aligned} \tag{4.7}$$

Again, note that the objective of (4.8) is directly taken from the second term of (4.5). $U(t)$ and the network loss probability can be obtained through regular feedbacks from the decoder. From (4.8), the choice of $f(t)$ is based on the video quality function and the current decoder buffer occupancy $U(t)$.

Notice that $U(t)$ plays a different role in each subproblem. A low $U(t)$ would cause $f(t)$ to increase due to it being a negative term in (4.8) and decrease $p(t)$ as it is a positive term in (4.6), while the opposite will happen when there is a high $U(t)$. This is precisely the kind of behavior we would like to have as a frame rate adjustment technique. We also defined an overflow control policy that ensures that the decoder buffer will not overflow, more details are in Appendix B.

5 Video Utility

Any frame and playout utility function could be used in our frame adjustment policy to represent $g(f(t))$ and $h(p(t))$. But it is desirable to have $g(f(t))$ and $h(p(t))$ as concave functions in order to provide tractable solutions for optimisation problems (4.6) and (4.8). This will mean that well known optimization tools such as gradient descent and Newton's method can be used to solve (4.6) and (4.8). In this section, we examine a particular choice of $g(f(t))$ and $h(p(t))$.

Recently, Ou et al [18] proposed video metric called VQMT that takes into account of both the frame and playout quality:

$$\begin{aligned}&VQMT(PSNR, p(t))\\&\quad= 0.928\left(1-\frac{1}{1+e^{q(PSNR-s)}}\right)\frac{1-e^{-b\frac{p(t)}{p_{max}}}}{1-e^{-b}}\end{aligned}\tag{5.1}$$

Where $PSNR$ is the sequence PSNR, q and s are constants and b is the modeling coefficient. Since we like to decouple into encoder and decoder objectives, we take the logarithm of (5.1):

$$\begin{aligned}&log(VQMT(PSNR, p(t)))\\&\quad= \log\left(0.928\left(1-\frac{1}{1+e^{q(PSNR-s)}}\right)\frac{1-e^{-b\frac{p(t)}{p_{max}}}}{1-e^{-b}}\right)\\&\quad= \log\left(0.928\left(1-\frac{1}{1+e^{q(PSNR-s)}}\right)\right) &(5.2)\\&\quad\quad+\log\left(\frac{1-e^{-b\frac{p(t)}{p_{max}}}}{1-e^{-b}}\right) &(5.3)\end{aligned}$$

From the above equation, we could use (5.2) for $g(f(t))$ and (5.3) for $h(p(t))$. However, to simplify (5.2) further, we also tried the non-logarithm form of (5.2) and (5.3), i.e.

$$\begin{aligned} & \mathit{Simple}\ VQMT(PSNR, p(t)) \\ & = 0.928\left(1 - \frac{1}{1+e^{q(PSNR-s)}}\right) \qquad (5.4) \\ & + \frac{1-e^{-b\frac{p(t)}{p_{max}}}}{1-e^{-b}} \qquad (5.5) \end{aligned}$$

We ran a simulation to compare Log VQMT with Simple VQMT by using them in (4.8). The results showed very little differences between Log VQMT with Simple VQMT. Therefore, we chose to use Simple VQMT for simplicity. More details on the comparison will be discussed later in Section 5.3.

Next, we investigate how to convert these into forms that will be suitable for $g(f(t))$ and $h(p(t))$.

5.1 Frame Utility Function

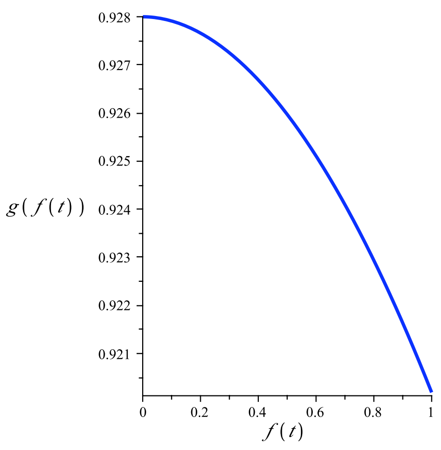


Figure 5.1: Frame utility function for football sequence. $f(t)$ is normalized by the default frame rate of 30 fps.

The frame utility function $g(f(t))$ should show the frame quality decreasing as the encoding frame rate $f(t)$ increases. This is because a higher $f(t)$ would result in lower average frame sizes, which will result in a lower average frame quality at a given available bit rate.

To begin with, it is well known that PSNR can be used to represent frame quality and can be modelled as [1]:

$$PSNR = a\log(r(t)) + c \qquad (5.6)$$

Where $a \geq 0$ and c are the model coefficients, while $r(t)$ is the frame size at time t. To use $f(t)$ as an input variable to this function, we set:

$$r(t) = \frac{ABR(t)}{f(t)} \qquad (5.7)$$

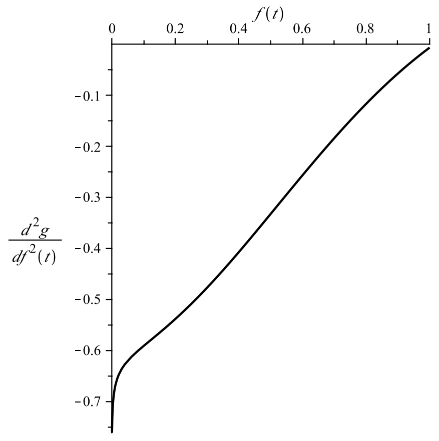


Figure 5.2: Second derivative plot for football sequence. $f(t)$ here is bounded and normalized by the default frame rate of 30 fps.

Where $ABR(t)$ is the available network bandwidth returned by the transport protocol. Eqn. (5.7) gives the *frame size* given the current $ABR(t)$ and frame rate $f(t)$. However, using (5.7) in (5.6) would cause it to be a non-concave function, making it difficult to maximize.

To obtain a concave function, we substitute (5.6) into (5.4) and use (5.7). We then get our proposed frame utility function (see fig. 5.1):

$$g(f(t)) = 0.928\left(1 - \frac{1}{1 + e^{q((a\log(\frac{ABR(t)}{f(t)})+c)-s)}}\right) \tag{5.8}$$

Eqn (5.8) is still not yet a concave function due to (5.4) being a sigmoidal function. However, it can be shown that (5.8) is concave if $f(t)$ obeys the following condition:

$$f(t) \leq ABR(t) \cdot e^{-\frac{1}{a}(\frac{\log(\frac{1+qa}{qa-1})}{q} - c + s)} \tag{5.9}$$

This upper bound on $f(t)$ can intuitively be viewed as the minimum acceptable frame quality. With this bound, we obtain a concave frame utility function (see fig. 5.2). The detailed concavity proof using this bound is presented in Appendix C.

To determine whether (5.9) produces bounds that are too low, we plotted the bounds over different $ABR(t)$. Fig. 5.3 shows the bounds for $f(t)$ calculated using (5.9) for two high motion CIF (352x288) sized sequences. Given that the sequences are in CIF resolution, the bounds produced are not too restrictive. While the bound of less than 15 fps at 200 kbps may seem too low, consider that the recommended *maximum* frame rate at 192 kbps is 7.5 fps for CIF sequences in the H.264/AVC standard [21]. Therefore, the bound is reasonably high enough and not too restrictive.

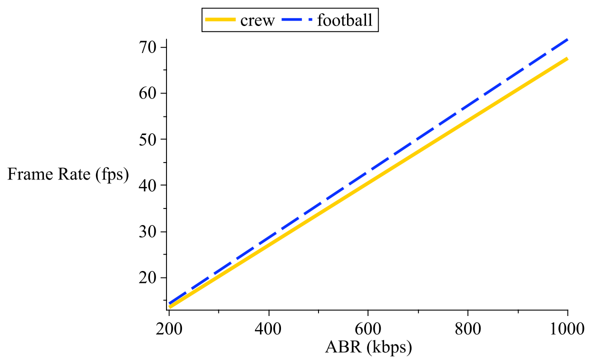


Figure 5.3: Upper bounds for $f(t)$ over different $ABR(t)$ for crew and football CIF sequence.

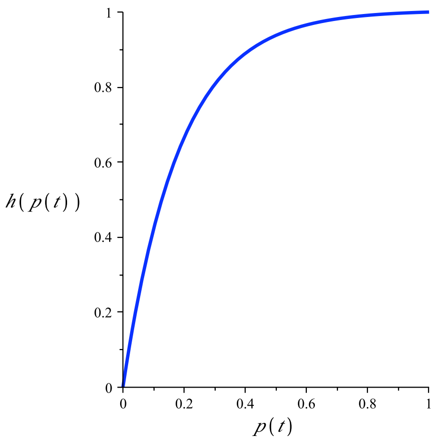


Figure 5.4: Playout utility function for football sequence. $p(t)$ is normalized to 30 fps.

5.2 Playout Utility Function

The playout utility function $h(p(t))$ should show the effect of the playout frame rate $p(t)$ on the user perceived smoothness of the video sequence at the decoder. A larger $p(t)$ would lead to a smoother sequence. For this, we directly use (5.5) (see fig. 5.4):

$$h(p(t)) = \frac{1 - e^{-b\frac{p(t)}{p_{max}}}}{1 - e^{-b}} \tag{5.10}$$

Where p_{max} is the maximum playout frame rate and $b \geq 0$ is a model coefficient. It can be verified by the second derivative test that (5.10) is a concave function (fig. 5.5), details of the concavity proof is in Appendix D.

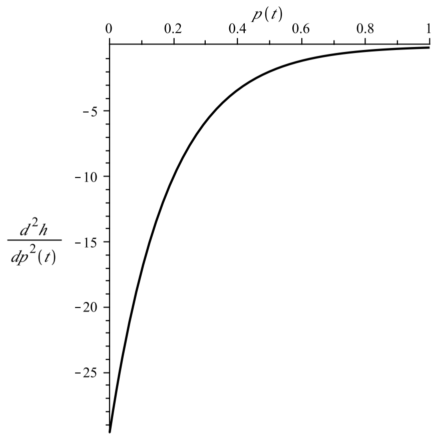
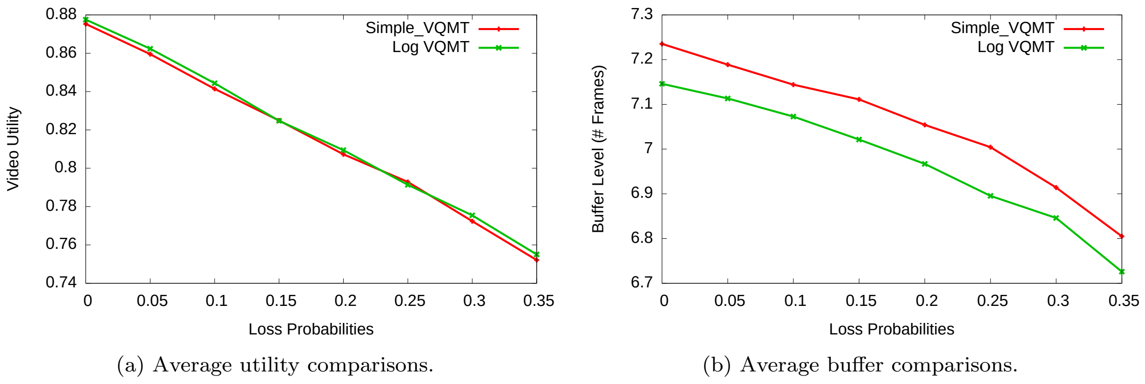


Figure 5.5: Second derivative plot for football sequence. $p(t)$ is normalized to 30 fps.



(a) Average utility comparisons.

(b) Average buffer comparisons.

Figure 5.6: Comparisons between Log VQMT and Simple VQMT.

5.3 Log VQMT vs Simple VQMT

Note that Log VQMT preserves the concavity of $g(f(t))$ and $h(p(t))$, this can be proved by using the second derivative test [3].

We ran simulations to compare the difference between Log VQMT and Simple VQMT. For the simulation, bandwidth is set to 1 Mbps and does not vary. We tested each scheme with an increasing number of simulated random losses of up to 35%, each loss scenario was repeat 10 times. Fig. 5.6 shows the results for each scheme. It can be seen that there is very little differences between the two schemes, which is why Simple VQMT was chosen.

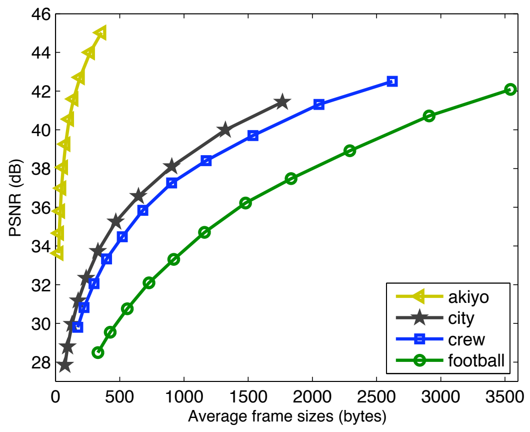


Figure 6.1: Frame size-PSNR graph for different sequences. Note that sequences with higher motion complexities need higher frame sizes to achieve similar PSNR of a low motion sequence.

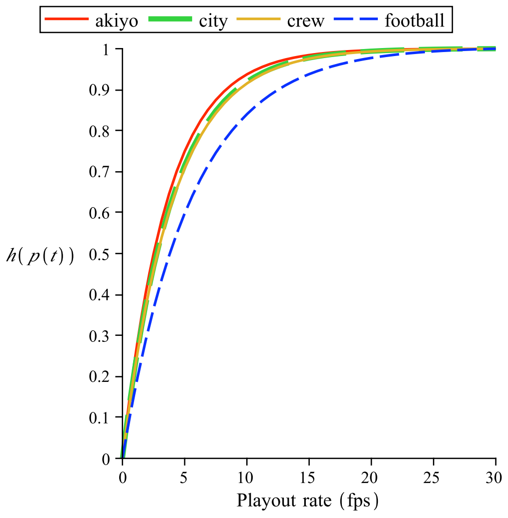


Figure 6.2: Playout utility $h(p(t))$ over different frame rate for the four sequences. Higher motion sequences tend to require a higher playout frame rate.

6 Simulation Results

6.1 Simulation Setup

We modified x264 [14] to perform our frame adjustment technique and tested it on four different CIF sized (352x288) sequences of different motion complexities, akiyo, city, crew and football. The order of motion complexity from the lowest to highest is, akiyo, city, crew and football (see fig. 6.1 and fig. 6.2). We set the constants of (4.8) and (4.6) as: $w_1 = w_2 = w_3 = 1$ $V = 7.5$ $\theta = 0.5$. The frame utility function $g(f(t))$ (5.8) is fitted for each sequence with $q = 0.34$.

The maximization of (4.6) and (4.8) is done using Newton's method [2].

The values of the model coefficient b from the playout utility (5.10) were taken from Ou et al [18], see Table 6.1 for the specific values. The playout delay of the decoder is set to 7.5 frames (250ms), and video packet arriving after the playout deadline is assumed to be lost.

Table 6.1: b for each sequence.

Sequence	b
akiyo	8.30
city	7.54
crew	7.38
football	5.43

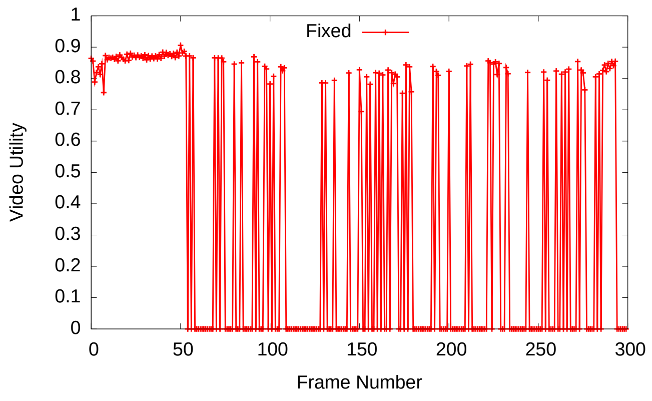
We used Ou et al's metric (5.1) to measure the received video quality as it takes into account the user's perception of both the frame and playout quality. We compared our technique with a setup that has both the encoding and playout frame rates set to 30 fps. In this section, our proposed buffer compensation framework will be called *LD*, while a setup with fixed encoding and playout frame rates will be called *Fixed*. *Fixed* simulates a conventional setup, such as [9,13], where the encoding and playout frame rates are determined beforehand. We also compared two variations of *LD*, *LD-Frame* which only optimizes the sending frame rate using (4.8) and *LD-Playout* which only optimizes the playout frame rate via (4.6).

We ran network simulations in NS-2 [17] and made use of the bandwidth and packet loss traces to simulate network transmissions. The transport protocol used was TFRC as it is one of the more popular TCP-friendly transport protocol with a smoothed bit rate pattern. Each network simulation was iterated 10 times to take into account of the randomness in packet losses and bandwidth fluctuations (see fig. 6.3 for an example of a single run). To test the effectiveness of our proposed technique in different network scenarios, we tested it in a wired network and in a wireless network.

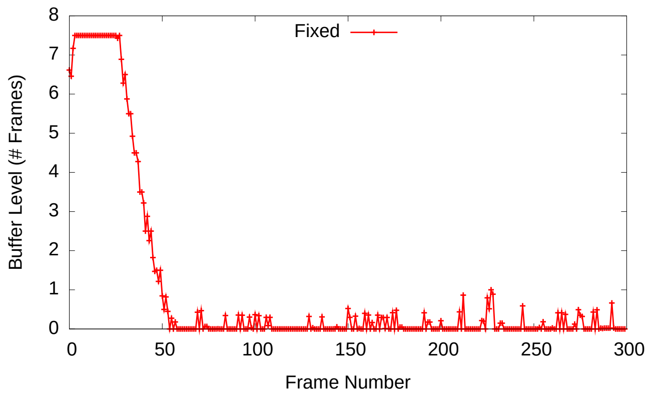
6.2 Wired Network

The aim of the wired network tests is to determine the performance under time-varying bandwidth fluctuations and delays due to network congestion. Packet losses in wired networks tend to be minimal and should not be a major factor here.

We used a simulated dumbbell network topology (see fig. 6.4) with competing TFRC flows for testing. The bottleneck link is set to 1 Mbps. We tested the different schemes through an increasing number of video flows in the network up to a maximum of 5 flows. The higher the number of flows, the higher the possible bandwidth fluctuations and delays will be. To introduce more network fluctuations, each test case also had Poisson traffic generated to simulate non-real-time background traffic. The results for these are shown in fig. 6.5.



(a) Video utility at each frame



(b) Decoder buffer levels at each frame

Figure 6.3: Results for crew in a wired network with 1 flow.

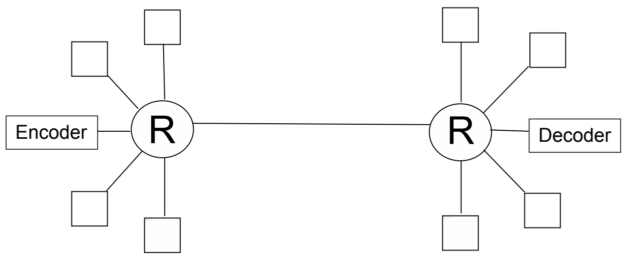


Figure 6.4: The simulated dumbbell wired network.

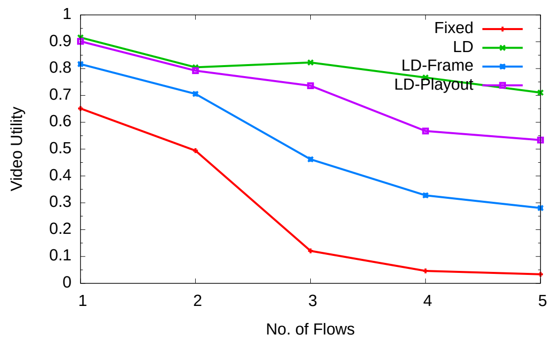
6.3 Wireless Network

We also simulated a wireless network with a central access point (AP). We assumed that a high bandwidth lossless link is connected from the AP to the decoder. Simulated wireless links are used to connect each individual node, which includes the encoder, to the AP. The 802.11 protocol was used for medium access control with the maximum rate set to 2 Mbps. Similar to the wired setup, we tested the schemes through up to 5 TFRC video flows.

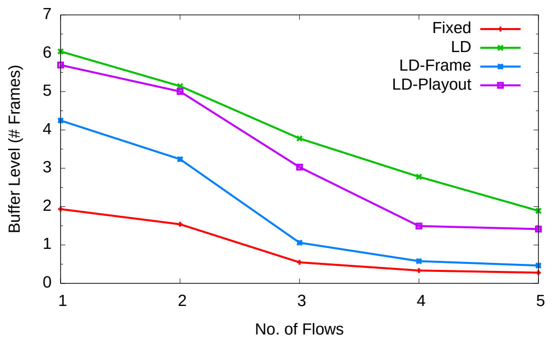
The wireless network setup introduces packet losses from channel conditions and medium contention in addition to the network congestion. This provides a much more challenging environment to provide constant video streaming. Fig. 6.7 shows the results for this setup.

6.4 Results Analysis

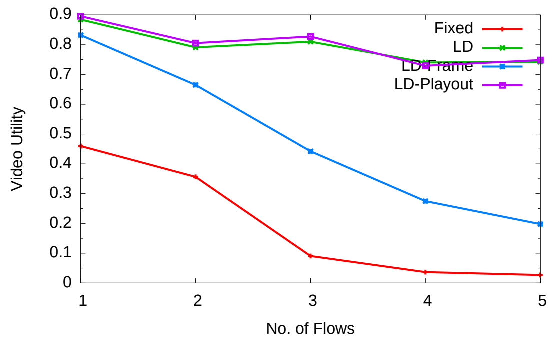
Overall, *LD* consistently outperforms *Fixed* in the four different sequences and in the wired and wireless network scenarios (see figs. 6.7 and 6.5). In fact, *LD* produced a video utility that is 70% higher than *Fixed* in one instance. It is also shown from the graphs that *LD* can handle the varying bandwidth as well as produce a more graceful degradation of video quality as the congestion and losses increases. Furthermore, the decoder buffer graphs show that *LD* maintained a buffer size that is less than 7.5 frames, thus maintaining the 250



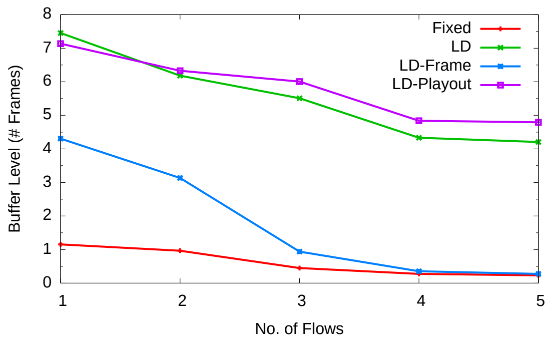
(a) Average video utility for akiyo



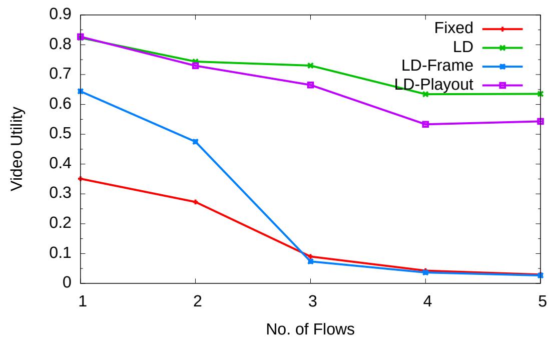
(b) Average decoder buffer levels for akiyo



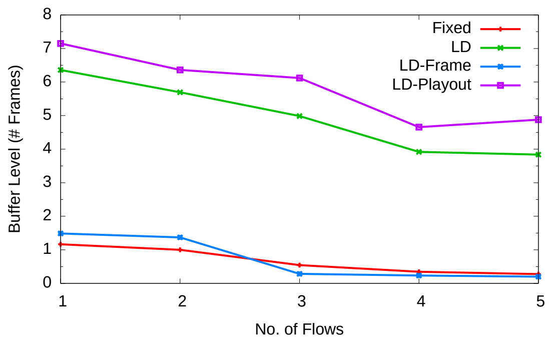
(c) Average video utility for city



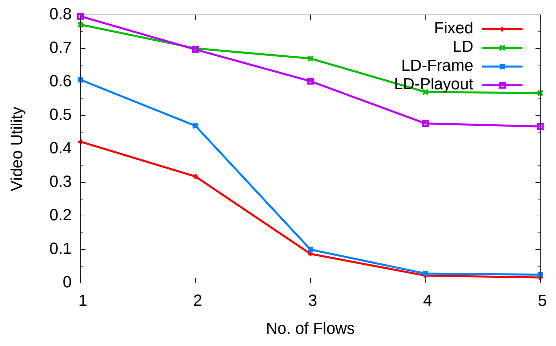
(d) Average decoder buffer levels for city



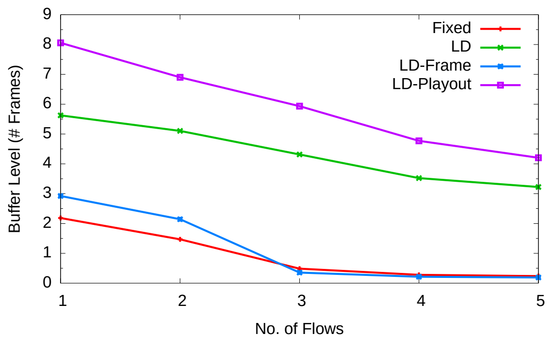
(e) Average video utility for crew



(f) Average decoder buffer levels for crew



(g) Average video utility for football



(h) Average decoder buffer levels for football

Figure 6.5: Wired network simulation results. Left column shows the average video utility graphs for each sequence. Right column shows the averaged decoder buffer levels for each sequence.

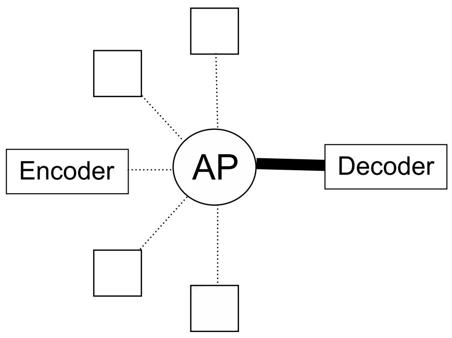


Figure 6.6: The simulated wireless network, dotted lines indicate wireless links, thick line is a lossless link to the decoder.

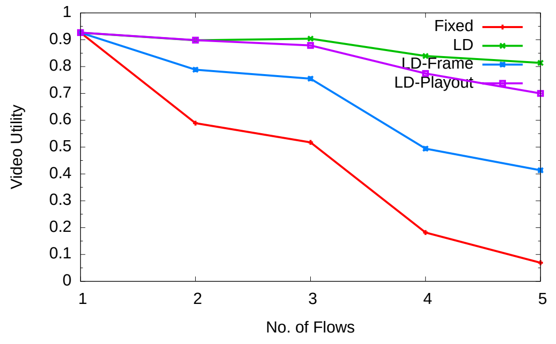
ms delay requirement. *Fixed*, on the other hand, produces rather low decoder buffer levels, which suggests that it is prone to decoder buffer underflows.

It is perhaps surprising that *Fixed* did not always achieve a good video utility even when there is only 1 video flow in the network. Fig. 6.3 shows a simulation run of crew sequence in a wired network that has 1 video flow and some background traffic. It is apparent from fig. 6.3 that *Fixed* produces good frame quality in this scenario, but the decoder buffer level kept dropping and never recovered. This subsequently resulted in a series of decoder buffer underflow events.

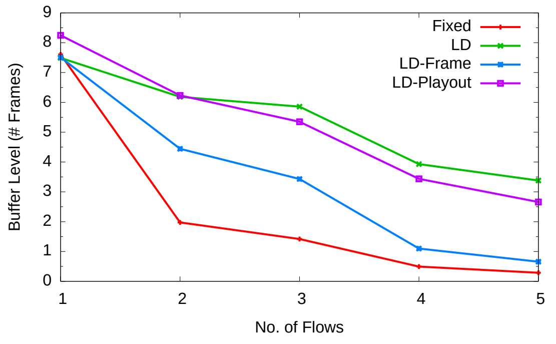
There are two main reasons as to why this effect occurs. Firstly, the typical rate controllers in the encoder produces a video stream that satisfies a *long term* averaged bit rate. As a consequence, in the *short term*, the rate controller may possibly produce a video stream that exceeds or under-utilizes the network bandwidth. Thus, it possibly requires more time to transmit a single frame than necessary and coupled with the bandwidth variations caused by background traffic, it causes the decoder buffer level to drop.

This then brings us to the second main reason, which is the motivation for this paper. In the *Fixed* scheme, a decoder buffer level drop will not be *directly* compensated. Possibly, the only way the decoder buffer will be compensated is when the rate controller somehow produces a stream that is of a lower bit rate than the network bandwidth. This might happen if the sequence shifts from a high motion scene to a low motion scene and the resulting frame sizes become *sufficiently* small enough. As it did not happen in these test scenarios, the decoder buffer level kept dropping and never recovered.

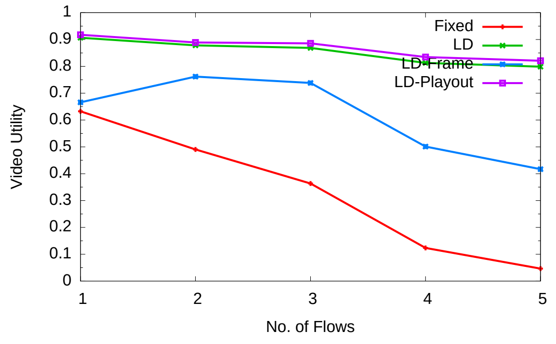
Comparisons of *LD* with *LD-Frame* and *LD-Playout* (fig. 6.7 and fig. 6.5) show that using a joint scheme of encoding frame rate and playout frame rate has, in most cases, reasonable performance improvements over a scheme that solely adjusts the encoding frame rate or only adjust the playout frame rate. It can be seen that *LD-Playout* tends to keep a higher buffer level, sometimes even exceeding the 250ms delay requirement. This is because the playout utility curves (fig. 6.2) mostly had little increase in utility from 15 fps onwards. As a result, *LD-Playout*'s optimization policy becomes more conservative, preferring to keep more frames in the decoder buffer to maximize its objective function (4.6).



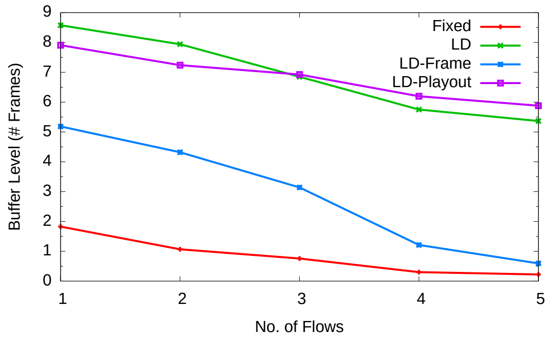
(a) Average video utility for akiyo



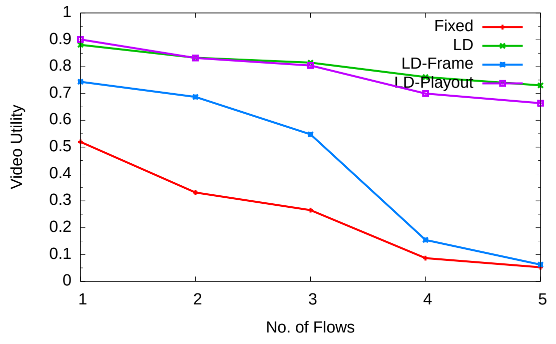
(b) Average decoder buffer levels for akiyo



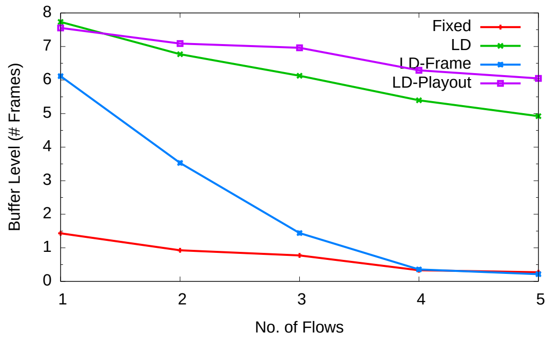
(c) Average video utility for city



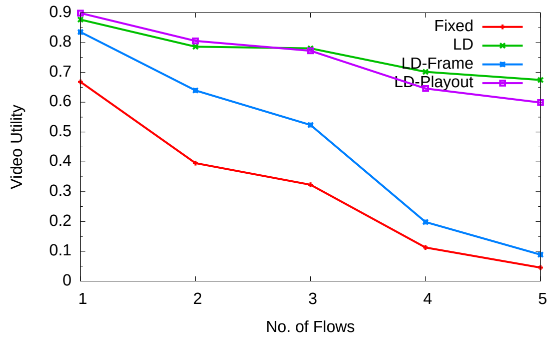
(d) Average decoder buffer levels for city



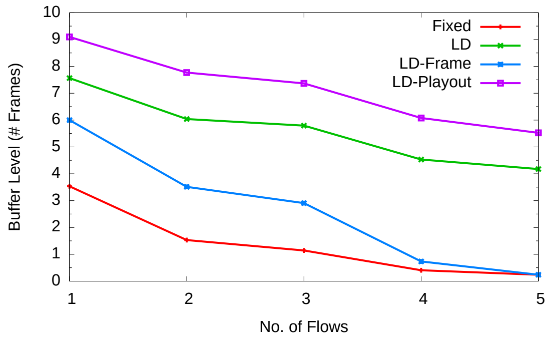
(e) Average video utility for crew



(f) Average decoder buffer levels for crew



(g) Average video utility for football



(h) Average decoder buffer levels for football

Figure 6.7: Wireless network simulation results. Left column shows the average video utility graphs for each sequence. Right column shows the averaged decoder buffer levels for each sequence.

7 Conclusion

We proposed a rate control framework that adjusts the encoding and playout frame rates to stabilize the decoder buffer. We derived optimization policies based on Lyapunov drift analysis. We then showed that the policies maximize the video quality both in terms frame quality and playout quality while stabilizing the decoder buffer. The proposed framework is shown to improve substantially over a standard setup of fixed encoding and playout rate.

Future work includes relaxing the assumption that $f(t) = \mu(t)$ and adaptively calculating the weighting parameters, w_1, w_2 and w_3 so that the performance is maximized and the frame rate change is smooth for different sequences.

Appendices

A Performance Bounds

Proposition 1. *The proposed frame rate adjustment policy stabilizes the buffer and achieves the following bounds (assuming $U(0) = 0$):*

$$U(t) \leq \Upsilon_{max} + f_{max} \tag{A.1}$$

$$O_{avg} \geq O^*_{avg} - \frac{B}{V} \tag{A.2}$$

where:

$$\Upsilon_{max} \triangleq \frac{Vw_2 g(f_{max}) + Vw_1 f_{max}}{2\theta f_{max}} \tag{A.3}$$

$$\begin{aligned} O_{avg} \triangleq \liminf_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} & w_1(F(f(\tau)) - p(\tau)) \\ & + w_2 g(f(\tau)) + w_3 h(p(\tau)) \end{aligned} \tag{A.4}$$

Proof of (A.1). This is proved by induction. When $t = 0$, (A.1) is satisfied given that $U(0) = 0$. For time slots $t \geq 0$, first lets assume that the current buffer state satisfies (A.1), i.e. $U(t) \leq \Upsilon_{max} + f_{max}$. To show that $U(t+1) \leq \Upsilon_{max} + f_{max}$, there are two cases to consider:

1. $U(t) \leq \Upsilon_{max}$. In this case, $U(t+1) \leq \Upsilon_{max} + f_{max}$, since f_{max} is the maximum number of frames that will reach the decoder in a time slot.
2. $U(t) > \Upsilon_{max}$. Here, based on the overflow control policy (see Appendix B), no frames will be sent to the decoder.

Therefore, $U(t+1) \leq U(t) \leq \Upsilon_{max} + f_{max}$. □

Proof of (A.2). Let $O(t) \triangleq w_1(F(f(t)) - p(t)) + w_2 g(f(t)) + w_3 h(p(t))$, then (4.5) becomes:

$$\begin{aligned} \Delta(U(t)) - V\mathbb{E}\{O(t)|U(t)\} \leq & \\ \theta B - 2\theta U(t)\mathbb{E}\{p(t) - F(f(t))|U(t)\} & \\ - V\mathbb{E}\{O(t)|U(t)\} & \end{aligned} \tag{A.5}$$

Suppose that $F(f(t))$ is strictly interior to the frame rates region Λ. This would mean for some $\epsilon > 0$:

$$F(f(t)) + \epsilon \leq p(t) \tag{A.6}$$

By combining (A.5) and (A.6), we get:

$$\begin{aligned}\Delta(U(t)) - V\mathbb{E}\{O(t)|U(t)\} \leq \\ \theta B - 2\epsilon\theta U(t) - V\mathbb{E}\{O(t)|U(t)\}\end{aligned} \tag{A.7}$$

From Neely et al [16], using (A.7), we can derive (A.2). □

B Overflow Control

To avoid buffer overflow, we also implement an overflow control policy. It can be seen that (4.8) needs to be positive for it to be maximized, specifically:

$$Vw_2g(f(t)) + Vw_1F(f(t)) - 2\theta U(t)F(f(t)) \geq 0 \tag{B.1}$$

By rearranging the terms, we get:

$$U(t) \leq \frac{Vw_2g(f(t)) + Vw_1F(f(t))}{2\theta F(f(t))} \triangleq \Upsilon(t) \tag{B.2}$$

Which we adopt as our overflow control policy. I.e., once $U(t) > \Upsilon(t)$, we set $f(t)$ to 0.

C Frame Utility Convexity Proof

Proposition 2. *The frame utility function $g(f(t))$ is a concave function if the following is satisfied:*

$$f(t) \leq ABR(t) \cdot e^{-\frac{1}{a}\left(\frac{\log(\frac{1+qa}{qa-1})}{q} - c + s\right)} \tag{C.1}$$

Proof of (C.1). In this proof, for notational convenience, we set $f \triangleq f(t)$ and $ABR \triangleq ABR(t)$. Now looking into the second order derivative of (5.8):

$$\frac{\partial^2 g}{\partial f^2} = -\frac{2q^2a^2\xi^2}{(1+\xi)^3 f^2} + \frac{qa\xi}{(1+\xi)^2 f^2} + \frac{q^2a^2\xi}{(1+\xi)^2 f^2} \tag{C.2}$$

Where:

$$\xi \triangleq e^{q(a\log(\frac{ABR}{f}) - c - s)} \tag{C.3}$$

It can be seen from (C.2) that for concavity, the following needs to be satisfied:

$$\begin{aligned}\frac{2q^2a^2\xi^2}{(1+\xi)^3 f^2} &\geq \frac{qa\xi}{(1+\xi)^2 f^2} + \frac{q^2a^2\xi}{(1+\xi)^2 f^2} \\ \frac{2qa\xi}{(1+\xi)} &\geq 1 + qa \\ \xi &\geq \frac{1+qa}{qa-1}\end{aligned} \tag{C.4}$$

Then substituting (C.3) into (C.4) and solving for f would yield (C.1).

□

Intuitively, since the encoding frame rate $f(t)$ has an inverse relationship with the frame quality, the upper bound on the encoding frame rate (5.9) can be seen as the minimum frame quality the encoder would allow.

D Playout Utility Convexity Proof

Proposition 3. *$h(p(t))$ as defined in (5.10) is a concave function.*

Proof.

$$\frac{\partial^2 h}{\partial p^2} = -\frac{b^2 e^{-b\frac{p(t)}{p_{max}}}}{p_{max}^2(1-e^{-b})} \tag{D.1}$$

Given that $b \geq 0$, $p(t) \geq 0$ and $p_{max} \geq 0$. It can be seen that (D.1) is always negative and thus (5.10) is a concave function.

□

Bibliography

[1] Cheolhong An and T.Q. Nguyen. Analysis of utility functions for video. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 5, pages 89–92, 16 2007-Oct. 19 2007.

[2] M. Avriel. *Nonlinear programming: analysis and methods.* Dover Pubns, 2003.

[3] S.P. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge Univ Pr, 2004.

[4] H.C. Chuang, C.Y. Huang, and T. Chiang. Content-Aware Adaptive Media Playout Controls for Wireless Video Streaming. *IEEE Transactions on Multimedia*, 9(6):1273–1283, 2007.

[5] M. Dai and D. Loguinov. Analysis of rate-distortion functions and congestion control in scalable internet video streaming. In *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 60–69. ACM New York, NY, USA, 2003.

[6] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP friendly rate control (TFRC): Protocol specification, 2003.

[7] L. Huang and M.J. Neely. The optimality of two prices: Maximizing revenue in a stochastic network. In *Proc. 45th Allerton Conf. on Communication, Control, and Computing*, 2007.

[8] M. Kalman, E. Steinbach, and B. Girod. Adaptive media playout for low-delay video streaming over error-prone channels. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(6):841–851, 2004.

[9] H.J. Lee, T. Chiang, and Y.Q. Zhang. Scalable rate control for MPEG-4 video. *IEEE Transactions on Circuits and Systems for Video Technology*, 2000.

[10] Mingzhe Li, Mark Claypool, and Robert Kinicki. Playout buffer and rate optimization for streaming over ieee 802.11 wireless networks. *ACM Trans. Multimedia Comput. Commun. Appl.*, 5(3):1–25, 2009.

[11] Y. Li, Z. Li, M. Chiang, and A.R. Calderbank. Content-Aware Distortion-Fair Video Streaming in Congested Networks. *IEEE Transactions on Multimedia*, 11(6):1, 2009.

[12] Y. Li, A. Markopoulou, J. Apostolopoulos, and N. Bambos. Content-Aware Playout and Packet Scheduling for Video Streaming Over Wireless Links. *IEEE Transactions on Multimedia*, 10(5):885–895, 2008.

[13] Z. Li, F. Pan, K.P. Lim, G. Feng, X. Lin, and S. Rahardja. Adaptive basic unit layer rate control for JVT. *JVT-G012-r1, 7th Meeting, Pattaya II, Thailand, Mar*, 14, 2003.

[14] L. Merritt and R. Vanam. x264: A high performance H.264/AVC encoder. *[online] http://neuron2.net/library/avc/overview_x264_v8_5. pdf.*

[15] L. Merritt and R. Vanam. Improved rate control and motion estimation for H. 264 encoder. In *Proceedings of ICIP*, volume 5, pages 309–312, 2007.

[16] M.J. Neely, E. Modiano, and C.E. Rohrs. Dynamic power allocation and routing for time-varying wireless networks. *IEEE Journal on Selected Areas in Communications*, 23(1):89–103, 2005.

[17] The Network Simulator NS-2. `http://www.isi.edu/nsnam/ns/`.

[18] Y.F. Ou, T. Liu, Z. Zhao, Z. Ma, and Y. Wang. Modeling the impact of frame rate on perceptual quality of video. *City*, 70:80–90.

[19] W. Simpson. *Video over IP: a practical guide to technology and applications.* Focal Pr, 2006.

[20] H. Song and C.C.J. Kuo. Rate control for low-bit-rate video via variable-encoding frame rates. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(4):512–521, 2001.

[21] G.J. Sullivan, P. Topiwala, and A. Luthra. The H. 264/AVC advanced video coding standard: Overview and introduction to the fidelity range extensions. In *SPIE Conference on Applications of Digital Image Processing XXVII*, volume 5558, page 53, 2004.

[22] E. Tan, Jing Chen, S. Ardon, and E. Lochin. Video tfrc. In *Communications, 2008. ICC '08. IEEE International Conference on*, pages 1767–1771, May 2008.

[23] J. Yan, K. Katrinis, M. May, and B. Plattner. Media-and TCP-friendly congestion control for scalable video streams. *IEEE Transactions on Multimedia*, 8(2):196–206, 2006.

[24] S. Yang and G. De Veciana. Bandwidth sharing: the role of user impatience. *GLOBECOM-NEW YORK-*, 4:2258–2262, 2001.

[25] Peng Zhu, Wenjun Zeng, and Chunwen Li. Joint design of source rate control and qos-aware congestion control for video streaming over the internet. *Multimedia, IEEE Transactions on*, 9(2):366–376, Feb. 2007.

[26] X. Zhu, E. Setton, and B. Girod. Congestion–distortion optimized video transmission over ad hoc networks. *Signal Processing: Image Communication*, 20(8):773–783, 2005.