2011-12-07

Quality Selection for Dynamic Adaptive Streaming over HTTP with Scalable Video Coding

Travis L. Andelin
Brigham Young University - Provo

Quality Selection for Dynamic Adaptive Streaming over HTTP with Scalable Video Coding

Travis L. Andelin

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Daniel Zappala, Chair
Sean Warnick
Robert Burton

Department of Computer Science

Brigham Young University

December 2011

ABSTRACT

Quality Selection for Dynamic Adaptive Streaming over HTTP with Scalable Video Coding

Travis L. Andelin
Department of Computer Science, BYU
Master of Science

Video streaming on the Internet is increasingly using Dynamic Adaptive Streaming over HTTP (DASH), in which the video is converted into various quality levels and divided into two-second segments. A client can then adjust its video quality over time by choosing to download the appropriate quality level for a given segment using standard HTTP.

Scalable Video Coding (SVC) is a promising enhancement to the DASH protocol. With SVC, segments are divided into subset bitstream blocks. At playback, blocks received for a given segment are combined to additively increase the current quality. Unlike traditional DASH, which downloads segments serially, this encoding creates a large space of possible ways to download a video; for example, if given a variable download rate, when should the client try to maximize the current segment's video quality, and when should it instead play it safe and ensure a minimum level of quality for future segments?

In this work, we examine the impact of SVC on the client's quality selection policy, with the goal of maximizing a performance metric quantifying user satisfaction. We use a combination of analysis, dynamic programming, simulation, and measurement to show that, in many cases, a client should use a diagonal quality selection policy, balancing both of the aforementioned concerns, and that the slope of the best policy flattens out as the variation in download rate increases.

Keywords: Video Streaming, SVC, DASH, Quality Selection Algorithms

ACKNOWLEDGMENTS

This work would not be possible without the assistance of several people. First, I thank my advisor, Dr. Zappala, for his help and expertise in the area of networking, especially with video protocols, experiments, and simulation. Much of our understanding of DASH came from its inventors at Move Networks, especially Drew Major, Paul Sherer, and Herrick Muhlestein, who tutored us on the workings of a commercial DASH implementation. In addition, I thank Dr. Warnick, Devon Harbaugh, and Vasu Chetty from the BYU IDeA Labs group. Their help was crucial for the development of the constant-rate optimal streaming proofs and the dynamic programming implementation. Also, I'm grateful to my parents for their encouragement and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The gradual evolution of the Internet over the past two decades has prompted the development of various methods for streaming video Begen et al. [2011], Wu et al. [2001]. Early on, video was streamed over User Datagram Protocol (UDP), a best-effort transport protocol that does not guarantee reliability or provide packet reordering. Initially, this proved to be an effective way to stream media, because UDP avoids the overhead of retransmitting lost packets. This is particularly useful for live or low-delay streaming, though it requires using a codec that can tolerate loss. However, UDP became less desirable for video streaming as NATs and Firewalls were introduced into the Internet, blocking the majority of UDP traffic. As a result, Transmission Control Protocol (TCP) became the prevalent method for streaming video, as it is the only common transport protocol that easily passes through NATs and Firewalls.

Gaining popularity among common methods of video streaming is a generic protocol known as *Dynamic Adaptive Streaming over HTTP* (DASH) Akhshabi et al. [2011]. DASH is a common way to stream high-definition (HD) video, and is used for both Video-on-Demand (VOD) and live streaming in software developed by Move Networks, Apple, Netflix, Microsoft, and Adobe Pantos [2010], Zambelli [2009]. DASH is adaptive in nature, where multiple quality profiles are available to the client. The client can select the most suitable video quality given the available network bandwidth at the time. After video is encoded into various quality levels, it is divided into two-second segments. Each segment is stored as a separate file. The client requests the individual video segments over Hypertext Transfer

Protocol (HTTP), and adapts the quality of the video relative to the available bandwidth provided by the network. The small duration of the individual video segments allows the player to dynamically switch between the different quality levels of a particular video.

1.1 Scalable Coding for DASH

Scalable Video Coding (SVC) is an extension to a the H.264/ MPEG-4 advanced video coding standard for video compression Li [2001]. With SVC, raw video can be split into multiple bitstreams, where each bitstream contains subsets of the raw data. Subset bitstreams can be recombined to additively increase the quality. SVC allows for raw video to be split by any of three different dimensions of quality: temporal resolution, spatial resolution, or SNR. Temporal resolution refers to the frame rate; by splitting temporally, bitstreams have reduced frame rates. Spatial resolution gives subset bitstreams reduced resolution, where they can be recombined to increase the overall resolution. With SNR, subset bitstreams have reduced quality; the signal-to-noise ratio of a bitstream measures quality loss of the subset bitstream when compared to the original video. A study done by Schwarz et al. [2007] shows that there is a 10% increase in the bitrate for spatial and SNR scalability.

The advantages offered by SVC clearly outweigh the overhead cost, and thus the use of SVC holds many promising improvements for DASH. Instead of splitting a video into different quality profiles, it can be split into subset bitstreams. These bitstreams can then be divided into two-second segments as before. In this way, rather than guessing the best overall quality for a segment before it is requested, the client can incrementally improve the quality of a segment by downloading additional bitstreams. Because a segment's quality can be increased anytime before its playback deadline, there are a large number of ways in which a video can be downloaded. Figure 1.1 shows a possible state in downloading a video. The blue blocks are those that have been downloaded, and the red blocks represent the candidates that can be requested next. We refer to the process of determining which block is selected next as the quality selection policy.

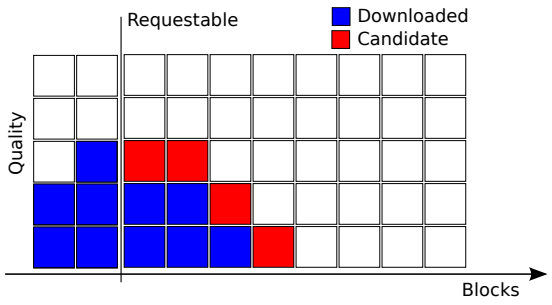


Figure 1.1: A DASH with SVC client must decide how to improve quality by downloading additional blocks of a video segment. A risk-tolerant client will download blocks for the current segment, and a risk-averse client will download blocks for future segments with low quality.

Of course, the quality selection policy will depend on the dynamics of the available bandwidth for the client. It is well known that the bandwidth a client can obtain will vary based on load, loss, and delay Mathis et al. [1997], Padhye et al. [1998], and is difficult to predict without knowing these factors. Thus the quality selection policy must balance the tradeoff between providing high quality for the current segment versus ensuring a consistent quality level for future segments. The client is essentially taking a gamble on the future rate it will achieve, with a risk-tolerant client downloading blocks vertically, improving current quality, and a more risk-averse client downloading blocks horizontally, ensuring future quality. The key question here is, *under which conditions should a client download with a diagonal policy, and how can it select the slope of this policy?*

1.2 Contributions

In this thesis, we pose DASH with SVC as a user-centric utility maximization problem. The solution to the problem is the quality selection policy that induces a playback that is pleasing to the the user. As this policy depends on the client's download rate dynamics, we first explore several constant-rate scenarios and show that different variations of a vertical policy are optimal, depending on the objective function. Next, we extend the problem to include independent, identically distributed (i.i.d.) rates over discrete time steps. We use dynamic

programming to find the optimal download policy for small-scale problems, where the video is short, and the rate distribution is a discrete random variable. With this more realistic rate modeling, we show that the optimal policy is diagonal, including alternating periods of prefetching (downloading out into the future at a lower quality) and backfilling (increasing quality close to playback). We employ network simulation to examine problems that are larger in scale and with a more general model for the rate distribution. We compare the performance of the above policies and their variations and show that as the variation of the rate increases, the best quality selection policy becomes diagonal, with a flatter slope needed to cope with increasing variability. Lastly, we conduct an experiment where PlanetLab rate measurements are used to justify the results of simulation and to validate the accuracy of rate models in their ability to predict which policy to use.

Chapter 2

Problem Formulation

We formally define the DASH with SVC quality selection problem, as the foundation on which we base our results. We seek to define our problem realistically as to ensure that our results be useful for real implementations.

2.1 Video Encoding

Consider a video consisting of N ordered segments, where each segment represents k seconds of video to be played. For each segment, the corresponding video data is encoded into b additive blocks. Denote s as the size of each block in the video, which we assume to be constant. (Typically the size of a block varies slightly with SVC coding.) For a video segment i, the presence of q_i of the b blocks allow for a video quality of q_i for that segment.

2.2 Client/Server Streaming Process

The video is stored on a server, and is downloaded by a video client across a network. The client downloads the video in a series of block selection steps. In a block selection step Σ, the client selects a segment, and then requests the next block for that segment. We denote the running history of the client's decisions at block selection step Σ by the vector $x^\Sigma \in \{0, 1, 2, ..., b\}^N$, where x_i^Σ corresponds to the number of blocks downloaded for segment i (0-indexed) by step Σ. With this said, the policy the client uses is a function

$$\pi : x \to u,$$

where $u \in \{0,1\}^N$ is a decision represented by the index vector with a 1 in the index of the segment to download for, and 0s elsewhere.

As video streaming is deadline-intensive, the client has a playback head that advances after every k seconds of downloading. Block selection begins at time zero, and continues until time kN. Receipt of a block for segment i before the playback deadline p_i results in an increase in q_i by 1; failure to meet this deadline means no change in quality. The playback deadline time for segment i is as follows, where i is 0-indexed:

$$p_i = (i+1)k$$

2.3 Network

Let the path between the server and the client be characterized by a delay d, which represents the time it takes for a message to travel from the client to the server, and back to the client. As delays typically have small variance, we assume d to be constant throughout the download. Let the rate behave as some stochastic process, and let w be the random variable which characterizes the rate distribution measured during each k-second interval (in blocks per segment.)

2.4 Objective

The user watching the video is delighted when the video quality is smooth and maintains high quality over time, and is particularly irked when the playback quality for any segment is 0. Such an occurrence means that the user must either wait as the client finishes downloading that needed block, or miss out on that k-second segment altogether. The user's happiness metric for the video can therefore be determined by the function y:

$$y(q,\lambda) = \left(\prod_{i=0}^{N-1} q_i\right)^{\frac{1}{N}} - \frac{\lambda}{N-1}\sum_{i=0}^{N-2}(q_i - q_{i+1})^2 \tag{2.1}$$

The left term is the *geometric mean* of qualities, which we refer to as the *quality score*. Considering all q for which $\sum_i^N q_i = j$, the choices of q for which the quality score is maximized are those q for which there is minimal variance, i.e., where all q_is are close as possible to $\frac{j}{N}$. Any zero element of q yields a quality score of 0. The right term is a penalty for *quality variation*, which is the change in quality over time. Note that this is not calculating variance; rather, this metric is measuring the amount that the video quality changes between adjacent segments. Here, λ is a nonnegative weighting parameter whose value determines how much the user values quality variation relative to the quality score.

With this problem formulation, the following fundamental question arises:
For a network with rate distribution w and delay d, a video of length N with b possible qualities, and a variation aversion factor λ, what policy will maximize the expected value of y?

Chapter 3

Optimality for Constant Available Rate

First, we take an introductory look at how to select blocks so as to maximize the objective given in Equation 2.1. In this section, we assume that the achievable download rate r is constant throughout the video, i.e., $P(w = r) = 1$. We assume that $d = 0$ to omit round-trip delay from consideration. Intuitively, it is not difficult to see that a vertical policy would do well, i.e., one where r blocks are downloaded for segment 0, then r blocks for segment 1, and so on until the end of the video. Here, we prove the optimality of several *vertical* policies that emerge in this case.

3.1 Preliminaries

Recall that $q \in \{0, 1, ..., b\}^N$ is the vector of qualities played back to the user during the video, where each $q_i \geq 0$. Let

$$\mu = \frac{1}{N} \sum_{i=0}^{N-1} q_i$$

represent the arithmetic mean of the total number of blocks that are successfully downloaded over the duration of the video. Recall that the first objective is to achieve high quality video by maximizing the geometric mean, which rewards consistent, high quality video, and to penalize any policy allowing a zero-quality. Let γ denote the geometric mean:

$$\gamma = \left(\prod_{i=0}^{N-1} q_i \right)^{\frac{1}{N}}$$

The relationship between arithmetic and geometric means is the well-known Arithmetic Mean – Geometric Mean Inequality:

Lemma 1. Arithmetic Mean–Geometric Mean Inequality. *Given the arithmetic mean of a set S consisting of N numbers, $\mu = \frac{P}{N}$, where the product of each element in S is given by $\alpha = \prod_{i=0}^{N-1} q_i$ and the sum of each element in S is $P = \sum_{i=0}^{N-1} q_i$, we show that $\mu \geq \sqrt[N]{\alpha}$, and equality holds if and only if $q_i = \mu = \sqrt[N]{\alpha}$, $\forall i$.*

From Lemma 1, we know that $\mu \geq \gamma$, and they are equal only when each element $q_i = \mu$. When μ is non-integral, i.e., $\mu \notin \mathbb{N}$, the closest that γ can be to μ is when elements of q_i are either $\lfloor \mu \rfloor$ or $\lceil \mu \rceil$.

The second objective is to minimize the variability in quality over time, since notable jumps between quality levels throughout the video are likely to degrade the user experience:

$$\sigma^2 = \frac{1}{N-1} \sum_{i=0}^{N-2} (q_i - q_{i+1})^2$$

3.2 Optimal Policies

We define the *Vertical* policy such that the client always downloads blocks for the current segment (the one immediately after the playback head). If the block being downloaded will not arrive on time, then a block is downloaded for the next segment.

Theorem 1. π^* for Constant Integral Rate. *If the achievable rate r is constant, and if this rate allows for download of an integral number of blocks in a time step, then the Vertical policy is the unique optimal policy π^*. In particular,the quality score is maximized and quality variation is 0.*

Proof. Because r is constant, the same number of blocks can be downloaded at each time step for N time steps, following from the assumption that the number of blocks the client can download is an integer value. Utilizing Lemma 1, we know that downloading μ blocks for the current segment during each time step maximizes γ. The quality variation σ^2 will be

minimized as it is zero. Therefore, the vertical policy is optimal when the rate is constant and μ is an integer.

Now we prove that the Vertical policy is the unique optimal policy for constant available rate. Consider any other policy besides the Vertical policy. The first time a block is downloaded for a segment other than the soonest available segment, the quality for that segment will be lower than μ at playback. Not only does this increase quality variation, but from Lemma 1, we know that this decreases γ, regardless of the quality of future segments attainable from a rate r. Hence, this policy is the only optimal policy in such a case. □

Next, consider consider the case when μ is not an integer. There is a tradeoff between minimizing the buffer size and minimizing variance.

Consider the case where μ is 3.5. Then the number of blocks downloaded at each time step by the vertical policy will be $x_{odd} = \lfloor \mu \rfloor = 3$ and $x_{even} = \lceil \mu \rceil = 4$. Although this policy maximizes the quality score, it clearly does not minimize σ^2.

We generalize this policy and call it *FloorVertical*. FloorVertical consists of downloading $\lfloor \mu \rfloor$ blocks in every time step, plus one extra block when enough extra bandwidth accumulates. Denote the extra bandwidth available during a segment as $\alpha = \mu - \lfloor \mu \rfloor$. Then for segment i, it downloads one extra block when $\lfloor \alpha i - \lfloor \alpha(i-1) \rfloor \rfloor = 1$. The following function denotes the number of blocks to download as a function of the segment:

$$FV(i) = \begin{cases} \lceil \mu \rceil & \text{if } \lfloor \alpha i - \lfloor \alpha(i-1) \rfloor \rfloor = 1 \\ \lfloor \mu \rfloor & \text{if } \lfloor \alpha i - \lfloor \alpha(i-1) \rfloor \rfloor = 0 \end{cases}$$

Theorem 2. π^* for Constant, Non-Integral Rate and $\lambda = 0$. *If the achievable rate r is constant and non-integral, and quality variation need not be minimized ($\lambda = 0$), but buffer space should be minimized, then the optimal policy π^* is FloorVertical.*

Proof. The quality vector q given by the FloorVertical policy is the range of the function FV. First, we show that no other valid choice of qualities can increase the quality score, and hence the score of the objective function is optimal. Consider the elements of q. All q_i are

one of two values: $\lceil \mu \rceil$ or $\lfloor \mu \rfloor$. Any other choice of qualities may be attained by perturbing elements of q; that is, by either pushing two elements in opposite directions away from μ, or by bringing two elements toward μ. From the proof of Lemma 1, it is clear that pushing two elements away from μ (e.g., moving a value of $\lfloor \mu \rfloor$ to $\lfloor \mu \rfloor - 1$ and increasing another element of q by 1) will lower γ. The other way to perturb q is to bring two elements toward μ (i.e., increasing $\lfloor \mu \rfloor$ by 1 and decreasing $\lceil \mu \rceil$ by 1.) This action has no effect on the quality score, as it merely rearranges elements of q. Thus, this policy is optimal in the sense that it achieves the maximal value of γ.

Now, we show that FloorVertical will also minimize buffer space in the sense that any downloaded block is stored for the minimum amount of time. This is true because this policy will always download a block for the current segment, never storing any for future use. Instead, any extra time in a segment is saved up until it is enough to download one extra block for a future segment. □

Note that FloorVertical will not minimize quality variation. To reach this minimum, we can ignore the buffering requirement and use extra time to download future blocks so that there is only one change in quality for the entire duration of the video. Continuing the example above, if the rate is 3.5, the client can play the first half of the video at quality 3 and the second half at quality 4. During the first half, the extra time resulting from downloading segments at the lower bitrate is used to download content for the second half of the video.

We generalize this policy and call it *MeanVertical*. Mean Vertical consists of downloading all completable blocks at a quality $\lfloor \mu \rfloor$, for $h - 1 = \lceil N\beta \rceil$ time steps, and using excess time to download at a quality of $\lceil \mu \rceil$ blocks starting at time step h, where $\beta = 1 - \alpha$, such that $q_i = \lfloor \mu \rfloor$, for $i < h$, and $q_j = \lceil \mu \rceil$, for $j \geq h$. See Figure 3.1 for an illustration.

Theorem 3. π^* for Constant, Non-Integral Rate and $\lambda > 0$. *If the achievable rate r is constant, and if this rate does not allow for download of an integral number of blocks in a time step, and we care about quality variation ($\lambda > 0$), then the optimal policy π^* is MeanVertical.*

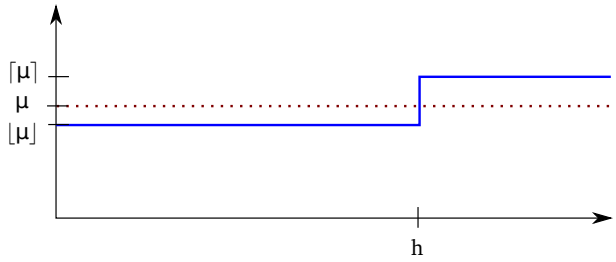


Figure 3.1: MeanVertical Policy: Optimal for Constant, Non-Integral Rate and Regarding Quality Variation

Proof. For reasons described in Theorem 2 and the fact that the optimal policy uses all available bandwidth, it maximizes the quality score. Since there is only one change in quality, the variation is at a minimum, $\sigma^2 = \frac{1}{N-1}$. □

Note that the maximum buffer size needed for the optimal policy is $\alpha\lceil N\beta \rceil$.

Chapter 4

Optimality for Variable Available Rate

Now that we have identified the optimal policies for a constant available rate, we look at the case where the client's download rate is variable. Dynamic programming Bertsekas [1995] is a powerful tool that can be used to find the optimal policy, π^*, when the rate follows a certain distribution, is independent and identically distributed (i.i.d.), and all decisions are made in discrete time. To find the optimal solution for such problems, we must alter the formulation so that it fits a general template for dynamic programming problems. We describe this auxiliary formulation, and then describe the implementation of the solver and corresponding results. The results validate the need for a diagonal quality control policy, alternating between periods of prefetching (downloading out into the future at a lower quality) and backfilling (increasing quality close to playback).

4.1 Dynamic Programming

Dynamic programming is a branch of control theory, an area of engineering and mathematics that can be used for solving problems with respect to a dynamical system. The purpose of control theory is to manipulate the inputs of a dynamical system to obtain the desired output. Figure 4.1 shows the diagram representing the quality selection control problem. Here, x represents the state, or the blocks that have already been received, and u represents the decision to make at each block selection step.

Dynamic programming is a method that can be used in a dynamical system to find the optimal policy, the decision u to make at each time step, given any feasible state x. The

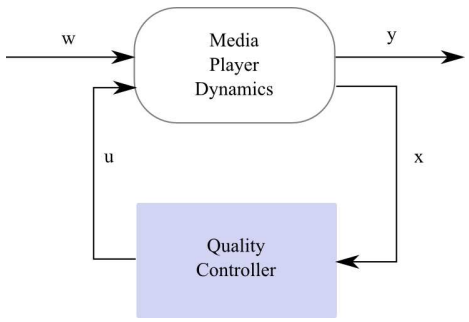


Figure 4.1: Diagram of the quality selection control problem. Given a rate distribution w, the controller seeks to maximize the expected value of y.

policy is optimal in that it maximizes the expected value of y. We maximize the expected value because the state x is influenced by the random variable w, and hence x becomes random in w. At each time step, a decision u is made, and it affects the state x, subject to random disturbance w.

Stated more formally, when given a random disturbance variable w, dynamic programming finds a function $\pi^* = (\pi_1^*, \pi_2^*, ..., \pi_n^*)$. Here, each π_i^* is a function $\pi_i : x \to u$, mapping any possible state x at time step i to the decision u. Following π^* at each time step will maximize the expected value of y. This function constitutes the optimal policy.

4.2 Auxiliary Formulation

To make the video streaming problem solvable with dynamic programming, we formulate an auxiliary problem as follows.

A video is encoded into N segments, and each segment is encoded into blocks of constant size. For simplicity, we assume no bound on b, the maximum quality for a segment. During each segment interval, the client makes m decisions, where each decision u specifies the segment in which to attempt a block download. Thus, the download consists of Nm time steps. Unlike the formulation in Chapter 2, decisions are made at fixed, discrete intervals, rather than after the previous block is entirely received. $u \in \{0, 1\}^N$ is an index vector, with a 1 in the position indexing the segment to download, and with 0s in every other position.

The random disturbance variable w is a distribution mapping the outcome (the fraction of a block received during that decision period) to its probability. Importantly, w is i.i.d across each decision stage. The client chooses the segment to download for at the beginning of a time step. At the end of the time step, it has received a fraction of a block according to w. The network has a round-trip delay d of zero.

The decision u, together with w, determine what gets added to the state x. x represents the cumulative quality received for each segment at any time. We refer to the subregion of x that is ahead of the current playback as the *active region*. The objective to this problem is the function $y(\lfloor x \rfloor, 0)$ from (2.1). We use the overloaded $\lfloor v \rfloor$ notation to denote the vector where each element of v is floored. Note that in this formulation, $\lfloor x \rfloor$ is equivalent to q in the formulation given in Chapter 2. The objective is computed after the last time step, and accounts for the overall quality of the video. For simplicity, we search for the optimal policy in terms of the geometric mean, omitting quality variation from consideration.

4.3 Implementation

The optimal policy π^* is straightforward to compute by moving backward in time. At the beginning of the last stage $(Nm - 1)$, we generate all possible states. From each possible state reachable by the last stage, we determine the decision that maximizes the expected value of y at the end of that stage. We remember this decision along with its expected value. Then, at the beginning of the second-to-last stage $(Nm - 2)$, we look at all possible states achievable by this stage. Again, we look at all possible decisions in this stage. Any of these decisions will land x into one of the possible states at stage $Nm - 1$, and for each of these we now have the optimal decision and expected value. The expected value for each state in $Nm - 2$ is found by summing the product of transition probabilities in w with the expected value of the corresponding end state. From this, we compute the optimal decision for each state in stage $Nm - 2$. We proceed in this fashion until stage 0. At this point, we have the optimal policy π^*.

N	m	Mem(GB)	Time(s)
3	55	3.9	138
4	21	3.9	160
5	11	3.7	161
6	7	3.3	160
7	5	2.8	153
8	4	3.5	150
9	3	2.3	150

Table 4.1: Approximate memory and time required to find solutions for various sizes of dynamic programming problems.

We implement the solver as a Java program. It takes as inputs N, m, and w. The output of the solver is a listing, by each time step, of the mappings from the current state to the block choice that is optimal at that particular time step.

Because this algorithm is exponential in space complexity, only small problems can be solved. Table 4.1 shows the approximate memory and time required to solve various problem sizes on a computer with an Intel Core 2 Duo 3.2GHz processor with 4 GB of RAM. Entries are the maximum values of m for each N that could be computed using this system.

In practice, it does not make sense to solve with fractional outcomes of w, as with HTTP, the client generally receives exactly what it requests beforehand. To make experiments realistic, we limit w to have outcomes of either 0 or 1 block per request, and vary the probability on each outcome. This creates a realistic distribution on blocks received per m attempts, or in other words, blocks received per segment interval.

4.4 Results

By examining the optimal solutions, we learn several key principles. First, we notice that the policy, although a function of the entire state x, is only influenced by the region of the state after the playback head. For example, if $N = 6$ and $m = 6$, and the current stage is 10, the playback is playing segment 2. The decision made at stage 10 is only influenced by blocks in the buffer or *active region*, in other words, blocks that are in segments 3-6.

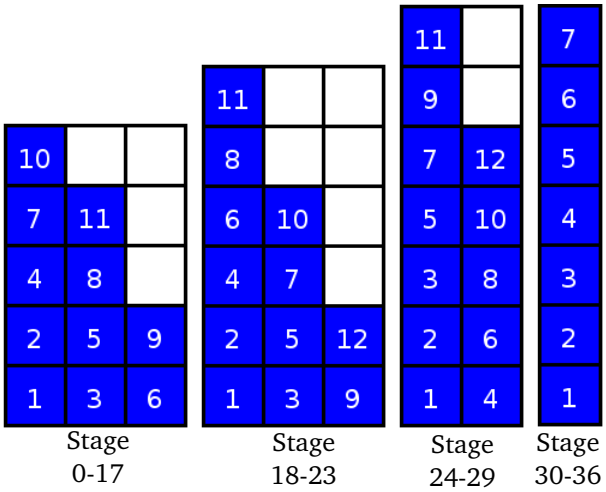


Figure 4.2: The optimal block selection policy at different stages of a video, where $N = 6$ and $m = 6$. $P(w = 0) = 0.5$, $P(w = 1) = 0.5$.

Another important finding is how the policy changes towards the end of the video. Figure 4.2 illustrates the block selection policy of a video where $N = 6$, $m = 6$, and w is such that $P(w = 0) = 0.5$, $P(w = 1) = 0.5$. Each grid gives an ordering, representing the sequence in which blocks are selected during different stages of the video download. For example, in the grid on the left, if the active region consists of block 1, then the optimal policy consists of selecting block 2. And if the active region consists of blocks 1 and 2, the optimal policy is to pick block 3, etc. Each subsequent grid to the right represents the optimal policy in future stages as it changes.

Note that this choice of w induces a somewhat normal distribution in the number of blocks received during each segment interval. The optimal policy for nearly the first half of the video is to download in a somewhat diagonal fashion; as the quality for immediate blocks increases, the policy will prefetch blocks in the future. Then, after some prefetching, blocks for nearer segments are again increased, and so on. As the video progresses, it is then optimal to choose blocks in a more "vertical" fashion. One obvious reason for this is that it cannot prefetch past the end of the video (e.g., in stages 30-36, it can only download for the next available segment.) However, in stages 24-29, instead of prefetching after the immediate quality is 2, prefetching begins after the immediate quality is 3.

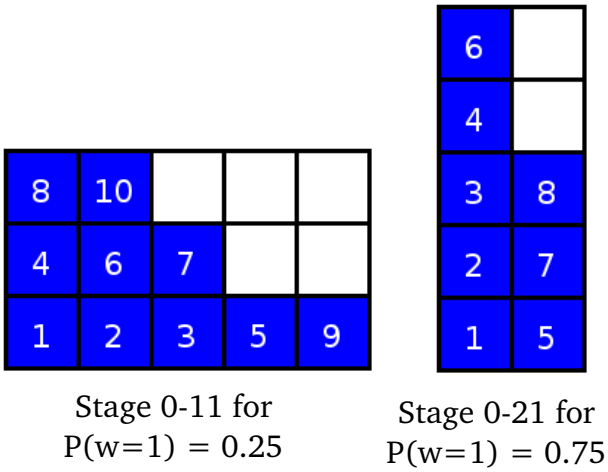


Figure 4.3: The optimal block selection policy at different stages of a video, where $N = 6$, $m = 6$. $P(w = 1) = .25$ (left) and $P(w = 1) = .75$ (right).

What happens if we change the probabilities in w? If $P(w = 1)$ is low, say 0.25, then the distribution over segment intervals has a low mean that is right skewed. If $P(w = 1)$ is high, say 0.75, then the distribution over segment intervals has a high mean that is left skewed. Figure 4.3 displays the optimal policies at the beginning of the video for each of these rate distributions. If the mean is low with a right tail, the optimal policy does more prefetching before increasing quality. If the mean is high with a left tail, the optimal policy is more vertical.

4.5 Discussion

Dynamic programming gives us useful information about policies that work well with different kinds of rate distributions; however, in reality there are several key differences. Because we can only solve small problems, perhaps there are aspects to the optimal policy for larger problems that are not apparent in smaller solutions. In addition, there are discrepancies between the auxiliary dynamic programming formulation and the formulation in Chapter 2. One difference is that the auxiliary problem has no notion of "running out of time" during a download; it merely understands the distribution on the number of blocks it can receive in a segment. The main formulation downloads blocks as previous ones are received. In addition,

the i.i.d. rate assumption is likely not realistic. In a wired Internet, the available rate is probably more likely to remain the same for longer periods of time.

Chapter 5

Simulation Study

To discover how these differences influence the success of the policies we have found thus far, we conduct a simulation study. The objective here is to model general bandwidth distributions and determine whether the advantages of a diagonal quality selection policy still hold.

Although available bandwidth in TCP over time has been modeled in terms of loss rates Mathis et al. [1997], Padhye et al. [1998], classifying typical distributions of available bandwidth in the wild seems to be an open question. This is an important issue for DASH with SVC, because effective quality selection depends on knowing the behavior of the available bandwidth. The rate model should encompass the dynamics of any underlying transport protocol, and does not need to be limited to TCP alone.

In this study, we use two rate models – the Truncated Normal and Markov Chain rate models. In the Truncated Normal rate model, we set a lower and upper bound on a normal distribution characterized by mean and variance, and choose a new rate from this distribution every k seconds. In the Markov Chain rate model, nodes represent rates, and arcs represent transition probabilities to new rates for each segment duration.

5.1 Policies

For our results to be insightful, we experiment on a large quality selection policy space. From the analytical work in Chapter 3 detailing the optimal policies under constant available rate, we use the *Vertical* and *MeanVertical* policies. The variable rate study using dynamic

programming in Chapter 4 motivates a class of *Diagonal* policies. A Diagonal policy will fill the active region so that blocks in it match a slope. Such a policy gives priority to the leftmost block needed to make the region match the slope. An example of a Diagonal policy with a slope of $-45°$ is found in the first 9 steps of the left grid in Figure 4.2. As the rate gets increasingly small and unpredictable (i.e., high variance), it would seem that the best thing to do is to have a purely *Horizontal* policy–one that would download the entire video in quality 1, then quality 2, and so on until the movie finished. In this case, the probability of having a zero quality segment would clearly be minimized. In our simulations, we compare the following policies:

- Vertical
- MeanVertical
- Diagonal with slopes varying between $-5°$ and $-85°$ in decrements of $5°$
- Horizontal

5.2 Simulator Design

To capture the details of the streaming setup in Chapter 2, we create an event-driven network simulator. We use this tool to measure performance of arbitrary policies under varying network conditions. Based on our problem formulation, it has the following inputs:

- Video length N, in segments
- Segment duration in seconds
- Network round-trip delay d
- Rate process
- Quality selection policy

The simulator's design includes the following four major components:

Client – The client manages the download of the video. It stores the video metadata, such as N, b, and segment duration. It has a downloader, which manages when to fetch the next block, and which blocks are currently stored in the buffer. The downloader works with the *policy manager* to determine which block to download. In addition, the client manages playback, keeping track of where the video is playing, and the quality level for each segment when it is played. The playback advances each segment duration period, and does not wait when the quality is zero.

Policy Manager – The policy manager interfaces with the downloader. Specifically, each available policy implements the function `int selectSegment(int beginSegment, Buffer buffer, Video metadata)`. Through this function, the policy takes as input the earliest available segment, the buffer (active region), and video metadata. The policy returns the segment corresponding to the next block to download.

Network – The network consists of a fixed round-trip time d and a rate manager that determines how the rate behaves over time. All rate managers implement the function `RateInfo getNextRate()`, which returns the next download speed (in blocks per second), and the duration for this rate (in seconds.) The rate managers used in the experiments reported here are the *Truncnormal* and *Markov* rate managers.

Core – The core makes up the "engine" of the simulator. It stores a priority queue of events, which it orders by event dispatch times. At runtime, the simulator operates by removing the event at the head of the queue, and dispatching it to appropriate component. Events that the core handles include `FETCH BLOCK`, `BLOCK RECEIVED`, `RATE CHANGED`, and `PLAYBACK ADVANCE`.

The client downloads a segment when it receives a `FETCH BLOCK` event from the core. Suppose it receives this event at time t. Since there is a round-trip delay of d, the client knows that the soonest it will receive that segment is at time $t + d$. Let n_{t+d} denote the segment closest to playback at time $t + d$. The client then calls `selectSegment`, with n_{t+d} as the `beginSegment`, and the current active region as the `buffer`. The policy then chooses

the segment to download. The client sends the corresponding request for this block to the network.

With this block request, the network calculates two times – the time the corresponding response will be received, and the best time for the client to send the next request. It then sends these events (`BLOCK RECEIVED` and `FETCH BLOCK`) to the core, where they are enqueued on the event queue. The response completion time is computed by looking at the delay d, and rates occurring after $t + d$. The best time to send the next request is one RTT before the response completion time. The client can estimate this time using RTT and rate measurements.

The playback head advances whenever the client receives a `PLAYBACK ADVANCE` event. The `RATE CHANGED` event is dispatched whenever the rate changes. The simulator runs the simulation based on its inputs, and gathers performance data. Among its outputs is the performance measure $y(q, \lambda)$.

5.3 Truncated Normal Rate

We implement a rate manager that creates a rate following the truncated normal distribution. The rate manager takes as inputs

- μ – the mean of the normal distribution representing rate
- σ – the standard deviation of the normal distribution
- a – the minimum outcome of the distribution
- b – the maximum outcome of the distribution
- *interval* – the duration for each generated rate

Observe that the truncated normal distribution has new moments (mean and variance) different from μ and σ. Define

$$d_a = \frac{a-\mu}{\sigma}$$
$$d_b = \frac{b-\mu}{\sigma}$$

Then

$$\begin{aligned} E(X|a<X<b) &= \mu + \frac{\phi(d_a) - \phi(d_b)}{\Phi(d_b) - \Phi(d_a)}\sigma \\ Var(X|a<X<b) &= \sigma^2\left(1 + \frac{d_a\phi(d_a) - d_b\phi(d_b)}{\Phi(d_b) - \Phi(d_a)}\right) \\ &- \sigma^2 E^2(x), \end{aligned}$$

where ϕ is the normal distribution's probability density function, and Φ is the cumulative distribution function.

We run an experiment with the following methodology. We fix a and b to bound the rate between 0 and 10 blocks per second. The *interval* is set to k, so that the rate varies every segment duration. We then pick 1,000 mean-standard deviation pairs (μ, σ) uniformly random in the intervals $0.25 \leq \mu < 6$ and $0 \leq \sigma < 3$. For each pair, we simulate all considered policies, where each policy gets simulated 200 times on a 1 hour video.

Figure 5.1 shows how the mean and standard deviation ($E(X)$ and $\sqrt{Var(X)}$) of the download rate determine which policy performs best when $\lambda = 1$. There is a region of moderate standard deviation where the MeanVertical policy performs best. The low mean, low-standard deviation region is dominated by the Horizontal policy, which outperforms others because of its few zero-quality segments. The high mean, high standard deviation region favors steeper Diagonal policies, and the moderate mean, high standard deviation regions are dominated by the more flat Diagonal policies.

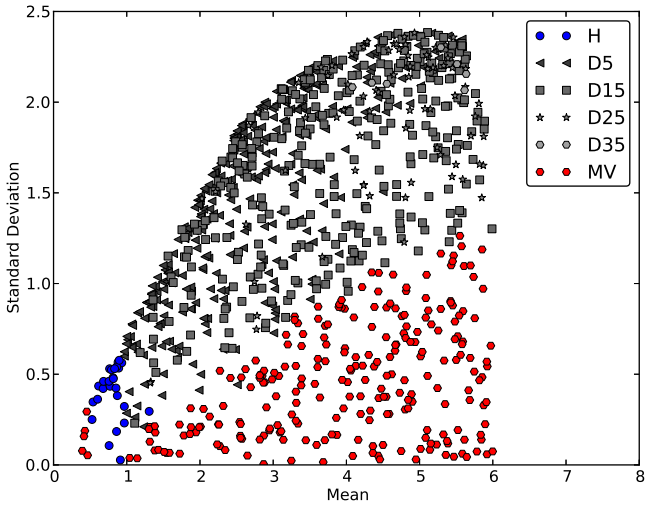
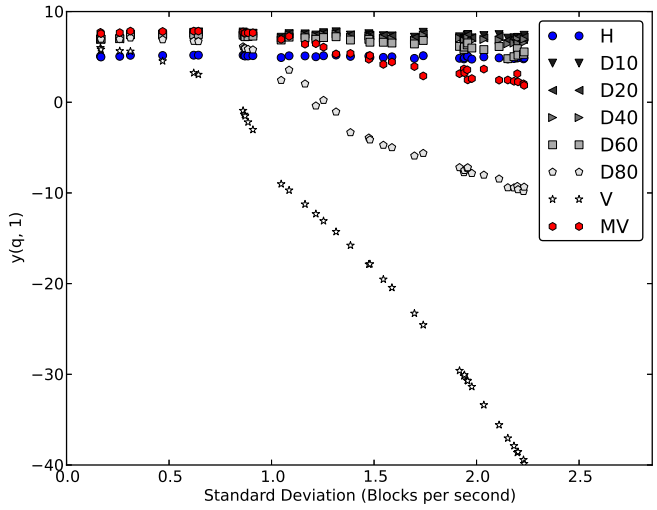


Figure 5.1: Best-performing policies under truncated normal rate model, and regarding quality variation ($\lambda = 1.$)

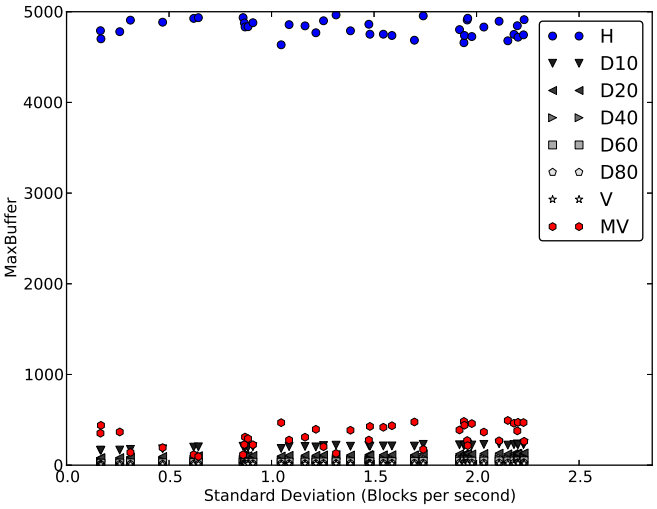
To see how each policy's score is affected by increasing standard deviation of the rate, see Figure 5.2, where $E(X)$ is between 3.75 and 4. Figure 5.2(a) shows the quality objective scores for the policies when $\lambda = 1$, meaning users equally value a high quality score and a low quality variation. Flat diagonal policies do the best in most cases, with MeanVertical having the best performance if rate variation is low. Note that the steeper Diagonal policies do well at first and then decline in effectiveness. Figure 5.2(b) shows the maximum buffer space used by each policy. The Horizontal policy uses much more buffer space than the other policies, since it always downloads the minimum quality for each segment before proceeding to higher qualities. Among the Diagonal policies, the buffer space used declines as the slope of the policy increases, but all of them use relatively small amounts of space.

5.4 Markov Rate

An important aspect of performance not captured in the Truncnormal rate model is the persistence and continuity of rates. In practice, rates tend to last a variable amount of time, sometimes much longer than one segment duration. Further, when rates change, the



(a) Quality objective where $\lambda = 1$.



(b) Maximum buffer space used, in blocks.

Figure 5.2: Performance of various policies where $3.75 < E(X) < 4$ and rate follows the truncated normal distribution.

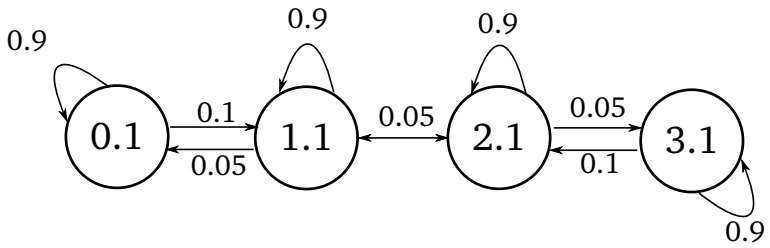


Figure 5.3: Markov chains such as this one can be used to model rate in a network. Arcs represent transition probabilities at each time step, and nodes represent rate (blocks per second).

change is somewhat continuous, e.g., a rate is more likely to change from 5Mbps to 4Mbps than from 5Mbps to 1Mbps. Therefore, it would be insightful to model rate persistence according to some other random process. To accomplish this, we use Markov chains Norris [1998] to model the rate. We let the nodes represent rates, and the arcs represent transition probabilities to adjacent nodes. See Figure 5.3 for an example of a Markov chain.

The Markov rate manager takes as input such a Markov chain. The rate selection is as follows. The simulator chooses a randomly selected starting node, representing the rate for the first segment duration. At the beginning of each subsequent segment duration interval k, the rate changes according to the transition probabilities in the graph. In this way, the Markov rate manager models networks where rates are persistent, and where future rates are highly dependent on current rates.

To show how mean and variance are found in a Markov chain, we first introduce some terminology. In a Markov chain with n nodes, the transition graph can be represented as an $n \times n$ matrix P, where element P_{ij} represents the probability of entering node j, given the current node is i. If the current node is i, then the probability that the current node will be j in k steps is

$$P_{ij}^k$$

If a Markov chain is regular, i.e., some power k of P has only positive elements,

$$W = lim_{k \to \infty} P^k$$

will have the same elements in each row. Let $\hat{w}$ be the common column of W, and further let the values at each node be characterized by the vector v, where $v \in \mathbb{R}^n$. Then the expected value is $E(P, v) = \hat{w} \cdot v$. Since $\hat{w}$ gives the probability distribution of being at each node, the variance is straightforward to compute from v and $\hat{w}$.

For the Markov rate model, we examine chains with different rates at each node, and vary the probability of remaining on a node. We consider chains where there are 7 nodes. Let $\alpha i + \epsilon$ be the rate of node i. Let each node have an arc to itself of probability $p : 0 < p < 1$, the probability of remaining on that node for another segment duration. Note that because $p \in (0, 1)$, this is a regular Markov chain. The remaining transition probability $(1 - p)$ is divided uniformly among adjacent nodes. For example, if node C has neighbors B and D, then node C will transition to node B with probability $\frac{1-p}{2}$, and to node D with probability $\frac{1-p}{2}$. Figure 5.3 shows a 4-node chain where $\alpha = 1$ and $\epsilon = 0.1$. For the Markov experiment, we vary α in the interval $[0.5, 2.5]$ in increments of 0.1, and vary p in the interval $[0.02, 0.98]$ in increments of 0.02. We set $\epsilon = 0.1$.

Each policy is simulated at each of these (α, p) points for a total of 200 simulated hours. Figure 5.4 illustrates the relationship between the remain probability p and the probability of staying in a state for s intervals. If $p = .95$, then there is a 40% chance that the rate will remain constant for about 20 segment durations.

Figure 5.5 shows which policies perform best for different expected values and different remain probabilities. From this figure, it is clear that low rates and high remain probabilities favor the Horizontal policy. Flat Diagonal policies are best when the remain probability is relatively high or the rate is low. Steep Diagonal policies do better as both the rate increases and the remain probability decreases. These are the cases where risk is minimized, because there is a lower chance of getting stuck with a bad download rate.

To see how each policy's score is affected by increasing the remain probability, see Figure 5.6, where $E(X)$ is equal to 3.75. Figure 5.6(a) shows the quality objective scores for the policies when $\lambda = 1$. The flattest Diagonal policy does best overall, while steeper

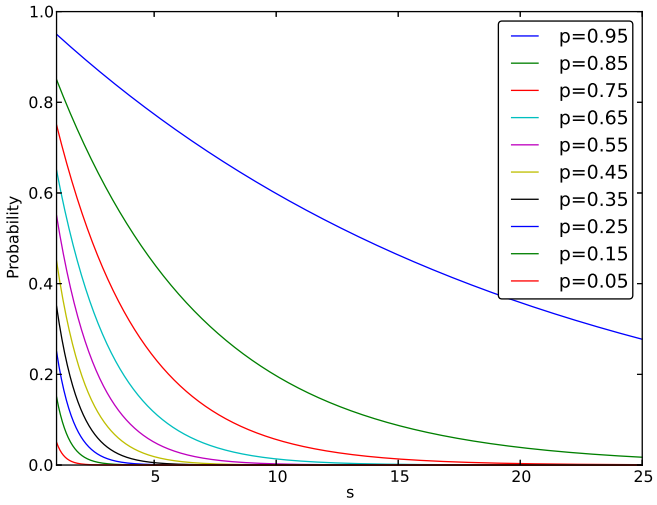


Figure 5.4: Probability of remaining in a state for s consecutive steps, given different remain probabilities.

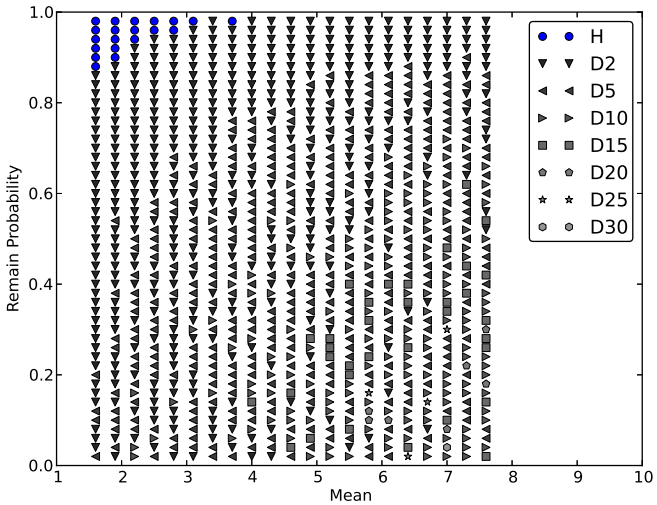
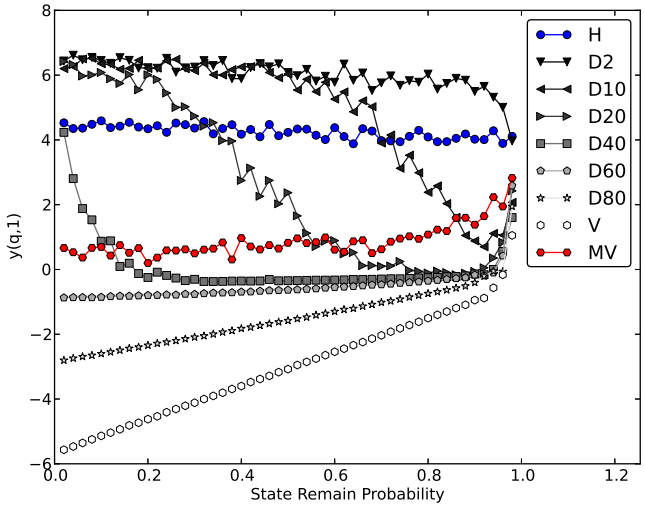


Figure 5.5: Best-performing policies under Markov chain model, and regarding quality variation ($\lambda = 1.$)

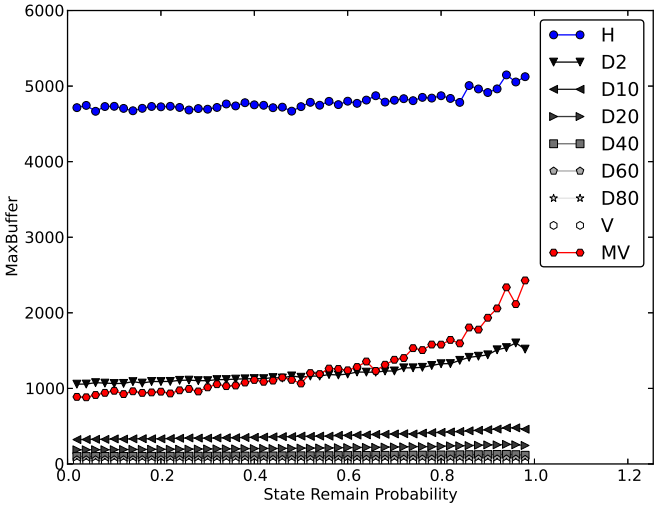
Diagonal policies begin to have a sharp drop in the objective as the remain probability increases toward 1. What is happening here is that the steeper policies take a greater risk, choosing to download more for nearby segments instead of future segments. As the remain probability increases, there is a greater chance that this was a poor choice, since the client now has a higher probability of getting stuck with a bad rate. The Horizontal policy clearly eliminates this risk. On the other hand, Figure 5.6(b) shows that the Horizontal policy has the largest buffer requirement. Even a slight departure from Horizontal, i.e., a very flat Diagonal policy, greatly reduces the buffer requirements while still doing well on the quality objective. Note that the Mean Vertical policy makes the opposite tradeoffs of the Horizontal policy, and the Diagonal policies balance quality with buffering.

5.5 The Effect of Round-trip Time

In the original problem formulation given in Chapter 2, the network has a round-trip delay d, which we have assumed to be zero in the mathematical analysis, dynamic programming formulation, and in the simulation results. Here we comment on the effect of d on the simulation results. Recall that the client is notified to request the next segment via a `FETCH BLOCK` event d seconds before it finishes receiving the currently downloading segment. When $d = 0$, then the simulator just requests the next block as soon as the current one is finished downloading. When $d > 0$, the simulator requests the next block *before* the current one is finished downloading, which negates any effect of propagation delay on the performance of the quality selection policies.



(a) Quality objective where $\lambda = 1$.



(b) Maximum buffer space used, in blocks.

Figure 5.6: Performance of various policies where $E(X) = 3.75$ and rate follows the Markov chain process.

Chapter 6

Rate Measurement Study

So far, we have only studied quality selection based on models of rate: Chapter 3 discussed a constant rate model, Chapter 4 introduced an i.i.d. m-stage outcome model, and Chapter 5 introduced the Truncated Normal and Markov Chain rate models. As results are highly dependent on the rate behavior, we had no basis to validate their accuracy. In this chapter, we discuss a measurement study, with rates measured from the Internet over TCP using PlanetLab. Based on the measured results, we analyze two things. First, we observe how well different policies perform on the measured rates. This will reveal the advantage of using a diagonal or horizontal policy over a simple MeanVertical policy. If this advantage is significant, then the measurement study justifies using different policies in practice. Second, we want to characterize the accuracy of different rate models. If the measured rates are representative of reality, then the more accurate a rate model is, the better it can predict which policy to use on an implemented client.

6.1 Methodology

To find a diverse sample of Internet rate measurements, we use *PlanetLab*, a group of computers located in various places around the world, used as a dedicated testbed for networking and distributed systems research. By transfering data between PlanetLab hosts, we find rate measurements subject to the dynamics of TCP and other factors in the Internet.

We randomly select 212 planetlab hosts from various locations around the world, including the United States, Canada, various European countries, Russia, China, and Aus-

tralia. To avoid high-bandwidth paths, we select at most one host per institution. On each host, we run an Nginx HTTP web server, which serves a video represented as a collection of 210KB blocks, each block stored as a file. A client also resides on each host, and at random times it selects a server to download the (virtual) video from. The client requests blocks from the server until it either receives the equivalent of a 1-hour video in full quality (3.8 GB), or until 1 hour of downloading is complete, whichever comes first. Each time it downloads a video, the client creates a log with the packet trace, containing the time each packet was received along with its size in bytes.

PlanetLab hosts have a bandwidth-control system which will adjust transfer speeds in a number of ways. Bandwidth is regulated by *shares*, *guarantees*, and *limits*. With fair shares, competing slices will receive a fraction of the available bandwidth depending on how many slices are sending on the host. If no slices are competing, all the available bandwidth is used. Shares will not significantly alter our measured network dynamics because TCP achieves fairness in a similar way. Guarantees are a lower bound on the rate a share limits to a slice; however, these do not alter our measurement because they are disabled by default. With limits, PlanetLab hosts are constrained to send less than 10GB in a 24 hour period. At 80% of this limit, sending rates are lowered significantly.

To overcome the effects of bandwidth limiting, one of the hosts acts as a dedicated tracking server that prevents nodes from sending more than 7.5GB within a 24-hour period. Each time a client initiates a video download, it queries the tracking server to find a valid server—one which has at least 3.8 GB available to send within the next hour. In this way, Planetlab hosts avoid any rate limiting, and rate dynamics are representative of the Internet. To minimize delay, the tracking server arranges client-server pairs to be within the same country, where possible.

Each host runs the *at* daemon to manage scheduling of video watching. Between video downloads, clients wait a random interval between 0 and 10 hours to avoid downloading too

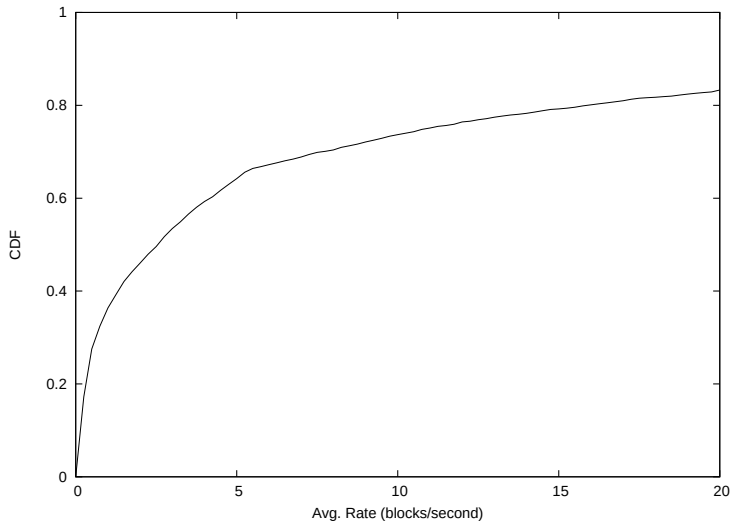


Figure 6.1: CDF of average rates in PlanetLab. Rates are measured from downloading a 1-hour video between random pairs of nodes.

frequently. Rate measurements were collected over a 7 day period from November 14 to November 21, 2011. During this time, 4,400 video traces were collected.

6.2 Results

Figure 6.1 shows the CDF of average client download rates in blocks per second. Download speeds ranged from a fraction of a block per second to several hundred blocks per second. We only consider the 2,204 traces where the average rate ranges between 0.6 and 15 blocks per second. A rate less than 0.6 is clearly too slow for video streaming, and average rates higher than 15 blocks per second would obviate the need for adaptive streaming.

Using the simulator described in the previous chapter, we test the set of policies (Vertical, MeanVertical, Horizontal, Diagonal with slope $-2°$, and $-5°$ to $-85°$ in decrements of 5) on each measured rate trace. Table 6.1 shows the frequency of best policies. While MeanVertical generally behaves well, the table indicates that there is a need to use different policies, depending on the rate characteristics.

Figure 6.2 shows the relationship of measured mean and standard deviation to the best-performing policy. To enhance clarity of the results, policies are partitioned into four

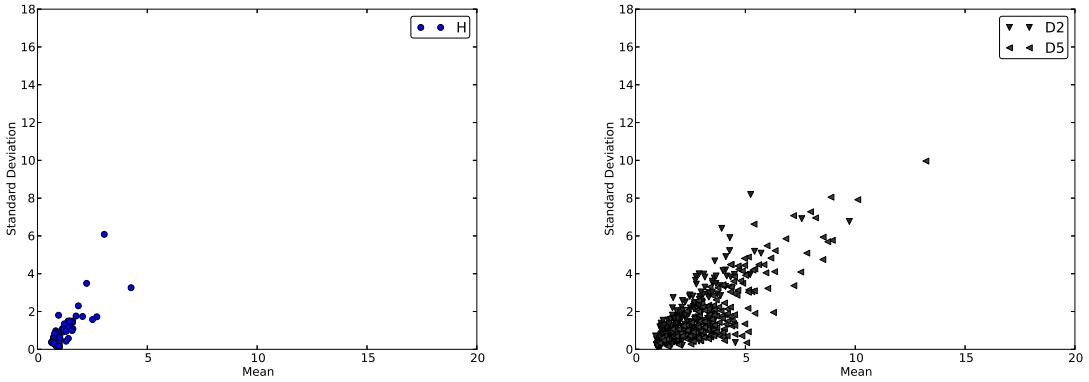
Policy	Frequency
MeanVertical	46.46%
D2	13.38%
Horizontal	11.89%
D5	9.44%
D10	6.22%
D15	4.63%
D20	2.77%
D25	1.50%
D45	1.32%
D35	0.64%
D40	0.18%
D55	0.09%
D85	0.05%

Table 6.1: Frequency of Best Policy from 2,204 traces with average rate between 0.6 and 15 blocks per second

sub-graphs. As is the case in Figure 5.1, the MeanVertical policy performs best over rates with low variance, and diagonal policies generally perform best when variance in rate is high. The Horizontal and flattest diagonal policies perform best where rates are very low, and steeper diagonal generally perform better where both variance and rate are high. The general rule for diagonal policies is that, as variance decreases and rate increases, the best slope to use becomes increasingly vertical.

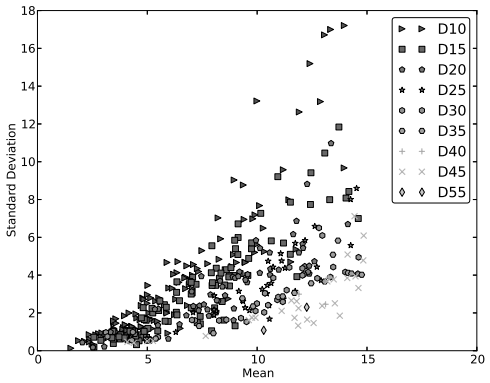
Figure 6.3 shows how well policies perform relative to each other in terms of quality and maximum buffer size in cases where the average measured rate is between 2 and 2.25 blocks per second. In Figure 6.3(a), multiple policies perform comparably to MeanVertical when standard deviation is low, and flatter policies perform better as it increases. Examining the maximum buffer size (Figure 6.3(b)) shows that the diagonal policies require much less buffer space.

Unlike the results from the clipped Gaussian rate model, the results from the measured rates are more scattered. This makes it harder to identify a clear relationship between mean, variance, and which policy to use. This is partly because the performance scores for each

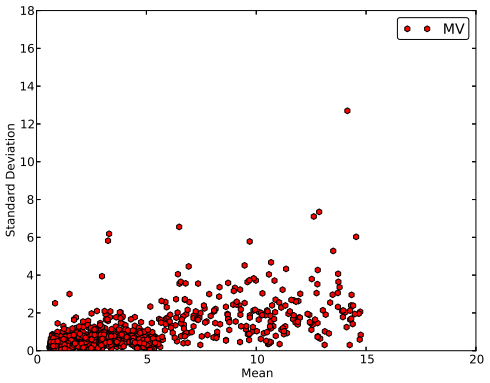


(a) Horizontal

(b) Flat Diagonal



(c) Moderate Diagonal



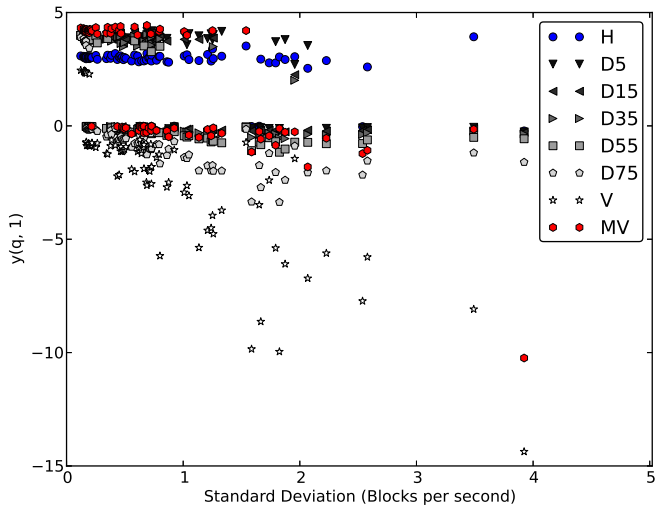
(d) Vertical

Figure 6.2: Best-performing policies under rates measured in PlanetLab, and regarding quality variation ($\lambda = 1.$) Policies are partitioned into subgraphs.

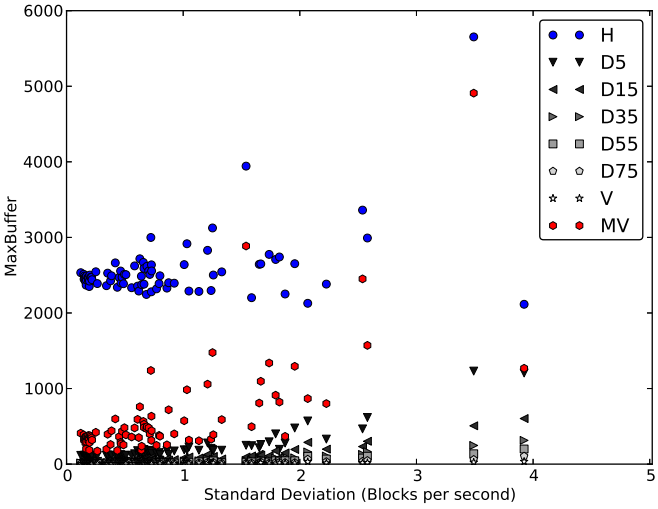
trace are not averaged over multiple repetitions. In addition, the mean and variance do not capture all essential elements of the rate as they relate to which policy to use.

6.3 Rate Model Evaluation

From the simulation study in Chapter 5, we see that, given a rate model, there is a mapping from its parameters (i.e., μ and σ with the truncated normal model) to the highest-performing policy, and the rate model coupled with a policy produces an expected value of performance. Until now, we have not validated the rate models against real Internet rate measurements.



(a) Quality objective where $\lambda = 1$.



(b) Maximum buffer space used, in blocks.

Figure 6.3: Performance of various policies where $2 < \mu < 2.25$ and rate is measured over PlanetLab.

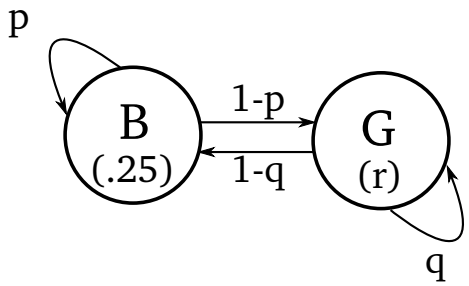


Figure 6.4: Two-State Markov chain used to model rate. This model consists of two states denoting bad and good rates, with transition probabilities between each.

Consider the learning problem that is associated with finding the best rate model to use. The client streams an hour-long video over the network, as discussed in Chapter 2. The rate can be measured in k-second time intervals, so denote $r \in \mathbb{R}^{\mathbb{N}}$ as the measured rate of the entire video. The client selects a policy π to use for the download, and the rate together with the policy produce the quality vector, and therefore the score for the video:

$$f(r, \pi) = y,$$

where

$$\pi \in \{MV, V, H, D_s\}, \qquad 0 < s < 90.$$

We want the client to accurately select from the set of available policies in order to maximize y:

$$F(r) = arg\max_{\pi} f(r, \pi)$$

The task for the client is to learn F.

In order for the client to learn F, it must accurately estimate f. However, the high dimension of r makes the function f difficult to learn. Because of this, a rate model is selected which has a much smaller input space, and it produces estimate $\hat{f}$ of f as has been done in Chapter 5. We characterize the accuracy of a model by the error of $\hat{f}$; thus, we can evaluate rate models by their ability to predict f. If a model does well at predicting f, it will also do well at estimating F.

With the ability to run the traces on the simulator, we can use the score y for each simulated policy as a benchmark to measure the accuracy of different rate models. We consider two models: the truncated normal model, and a 2-state Markov chain.

The latter is used because Markov chain models can capture aspects of smoothness and persistence in measured rate that i.i.d. models, such as the truncated normal, do not. The 2-state Markov chain shown in Figure 6.4 has 2 states, B and G. State B represents a low (bad) rate of 0.25 blocks per second, and state G represents good rate of r, where $r \geq 0.5$ blocks per second. Note that we assume $k = 2$, where a good rate must be at least 1 block per segment interval. This model captures expected rate, persistence, and smoothness with only 3 variables: p, q, and r. This 2-state chain is similar to the Gilbert model used by Bolot et al. [1999] for adaptive Forward Error Correction control.

We evaluate the accuracy of each model based on the disparity between trace and model performance. For each trace, we estimate parameters for models, then simulate the trace, Gaussian, and Markov rates with each policy. The performance from each model is compared with trace performance, and thus error is computed. To ensure significance of the performance from the rate models, we simulate each model 20 times with the estimated parameters, and record the average performance.

The Truncated Normal model estimates parameters by computing the sample mean and standard deviation from the trace, and assumes a minimum rate of 0 and a maximum rate of 20 blocks per second. The 2-state Markov chain model calculates r from the average of all rates higher than 0.5 blocks per second, and computes state transitions from the probability that the trace bandwidth will alternate states over a 2-second interval. For example, p is computed as the number of times (2-second intervals) that the bandwidth remains in state B divided by the total number of times the bandwidth is in state B.

Table 6.2 shows the Root Mean Squared (RMS) error of both models by policy. For every policy excepting for Vertical, the Markov model predicts the performance of the policy with greater accuracy than the Truncated Normal model, sometimes by a factor of 2. Both

models can predict with greater accuracy on policies with flatter slopes. These results show the advantage of using Markov Chains to represent Internet rates when smoothness and persistence matter.

Policy	**Gaussian**	**Markov**
MeanVertical	6.45	4.78
Horizontal	1.73	1.40
D2	2.59	2.00
D10	3.32	2.30
D20	3.87	2.39
D30	4.34	2.43
D40	4.65	2.48
D50	5.24	2.89
D60	5.59	2.92
D70	6.10	3.13
D80	6.94	3.88
Vertical	41.15	42.39

Table 6.2: RMS Error of Model Performance by Policy

Chapter 7

Related Work

DASH protocols are playing a major role in streaming video on the Internet today. The most widely used are Apple's *HTTP Live Streaming*, Microsoft's *Smooth Streaming*, *Move Networks* , and *Netflix*. DASH was pioneered originally by Move Networks, then subsequently adopted by Apple and Microsoft. Live Streaming is currently used both in devices using IOS and also in QuickTime. Smooth Streaming is built into Microsoft Silverlight. The Netflix video client is also implemented in Silverlight, but has several differences from Smooth Streaming in how it is implemented. In addition, Adobe has a DASH variant in their Open Source Media Framework (OSMF).

Two reasons motivate the use of HTTP in DASH protocols. First is the ease of deployment in servers and caches. HTTP is already implemented in ordinary web servers and caches, and many adaptive client variants use nothing more than HTTP for communication. Second, clients can make use of the *Byte Range* requests and *Pipelining* capabilities of HTTP. Byte Range requests can be used to request specific frames when the user seeks (i.e., fast-forwards or rewinds); both the Move and Netflix clients do this. The Move also client uses Byte Range requests to subdivide the download of a two-second segment between multiple parallel TCP connections. The Pipelining functionality allows a TCP connection to request subsequent segments before the currently downloading segment is received. We noted earlier that DASH protocols typically encode videos into two-second segments. Netflix departs from this convention; it instead requests constant-length byte range requests on the original video.

An in-depth study evaluating the performance of adaptive streaming over HTTP was performed by Akhshabi et al. [2011]. They evaluate two commercial streaming protocols Microsoft Smooth Streaming and Netflix, along with Adobe's open source adaptive player. Simple experiments are run over a local network using a DummyNet router. They seek to answer (1) how a video player reacts to changes in underlying bandwidth, (2) what occurs when two players compete for bandwidth under the same bottleneck, and (3) how well adaptive streaming performs with live content. They identify issues in several implementations with stability, convergence, and fairness.

Another study, by Fecheyr-Lippens [2010], conducts an experiment comparing existing multimedia streaming systems with Apple's HTTP Live Streaming. Specifically, he looks at client CPU load, client memory consumption, and server capacity of Adobe Flash and RTSP streaming systems, comparing it with Live Streaming. Further, he gives details on OS compatibility with each streaming systems, and instructions on how to set up each system. His report is a useful starting point for deploying an experiment with Apple's Live Streaming.

Although we have not yet seen commercial implementations of DASH with SVC, academic researchers have recently explored this idea. Abboud et al. [2011] examine the use of SVC in the context of peer-to-peer video-on-demand streaming. Their paper examines the impact of different quality adaptation algorithms on performance. A similar video encoding is used. They divide the buffer into a high priority zone and a low priority zone. The high priority zone constitutes segments close to the playback, and the low priority zone is the segments further out. As soon as the requirements for the high priority set are met, prefetching begins on the low priority zone. Sánchez de la Fuente et al. [2011] show how HTTP cache efficiency can be improved when combining SVC with DASH. A video streaming system often deploys caches at the ISP level to decrease transmission delay and improve scalability of a video streaming system.

The idea of using scalable video coding to adapt the quality of a video while streaming it over the Internet is not a new idea. A seminal work in this area is the paper by Rejaie et al. [2000]. This paper designs a quality adaptation mechanism for the server that adds and drops layers of the video stream to adjust to long-term trends in the available rate on the connection. Their protocol is designed to be used over UDP with a TCP-friendly rate control mechanism that reacts to congestion on shorter time scales.

Chapter 8

Conclusion

In this work, we have studied the relationship between the characteristics of available bandwidth and the performance of different quality selection algorithms when DASH is used with SVC. It is clear that using a scalable codec offers enhancements not provided by traditional DASH architectures.

8.1 Contributions

First, we give a concise formulation for the video streaming problem involving adaptive streaming over HTTP using a scalable codec, and then pose it as a utility maximization problem from the perspective of the user. Such a formulation serves as a framework for modeling performance.

Second, we derive optimal policies for constant and variable available bandwidth. Under constant rate, the policies are somewhat trivial, as they are essentially vertical in nature. While Vertical and MeanVertical policies are intuitive for constant available rate, Diagonal policies outperform vertical policies when rate is highly variable and the mean rate is low. We find justification for Diagonal policies in the optimal solutions to our dynamic programming formulation under a variable rate distribution.

Third, we evaluate the performance of different quality selection algorithms under two different rate models – truncated normal rate and Markov chains. From the truncated normal rate model, we observe that a Diagonal policy performs well when the rate is highly variable. When the mean rate is lower, or the standard deviation is high, the more flat

Diagonal policies tend to perform best. With the Markov chains model, we find that when rates are persistent with the possibility of being very low, the use of flat Diagonal policies or the Horizontal policy tends to work best.

Finally, we collect rate measurements from PlanetLab, and use them as a benchmark for performance and rate model evaluation. The results from the measurement study confirm the results from the simulation study–that diagonal policies outperform the MeanVertical policy when rate variation is high, and that there are performance and storage benefits from using different policies. Also, results indicate that the 2-state Markov chain rate model can predict the performance of the various policies with higher accuracy than the truncated normal rate model.

The benefits we have observed from Diagonal and Horizontal policies advocate the use of scalable coding in DASH algorithms.

8.2 Future Work and Open Questions

The focus of our study has been on quality selection; we omit several implementation-level optimizations to video streaming. One option is to develop a rate controller that could alter the number of TCP connections to achieve a desirable rate. In Ferragut and Paganini [2010], Ferragut and Paganini study how to achieve desired rates from a user-centric perspective through opening multiple TCP connections. In addition, Kuschnig et al. [2010] find that opening multiple TCP connections will have a desirable affect on reducing the amount of bandwidth fluctuation. The Move Networks DASH client opens 3 parallel connections and uses byte range requests to download a segment in parallel.

Our study indicates that the precise diagonal slope that works best depends on the download rate variation. Future work could develop a dynamic diagonal policy that adjusts the slope based on measurements of the standard deviation of the download rate. Because many devices have limited memory capacity, it would be interesting to explore how perfor-

mance of the various policies is affected when capacity is limited. It would also be interesting to explore how to design quality selection algorithms that ensure good startup performance.

This work also suggests that it will be important to develop models of the available bandwidth an application can expect, and in particular how this bandwidth changes over time. These models can be used to help make more intelligent quality selection choices.

References

O. Abboud, T. Zinner, K. Pussep, S. Al-Sabea, and R. Steinmetz. On the Impact of Quality Adaptation in SVC-based P2P Video-on-Demand Systems. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, pages 223–232, New York, NY, USA, 2011. ISBN 978-1-4503-0518-1. doi: http://doi.acm.org/10.1145/1943552.1943582. URL `http://doi.acm.org/10.1145/1943552.1943582`.

S. Akhshabi, A. Begen, and C. Dovrolis. An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, pages 157–168, New York, NY, USA, 2011. ISBN 978-1-4503-0518-1. doi: http://doi.acm.org/10.1145/1943552.1943574. URL `http://doi.acm.org/10.1145/1943552.1943574`.

A. Begen, T. Akgul, and M. Baugher. Watching Video over the Web: Part 1: Streaming Protocols. *Internet Computing, IEEE*, 15(2):54 –63, 2011. ISSN 1089-7801. doi: 10.1109/MIC.2010.155.

D.P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific Belmont, MA, 1995.

J.-C. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive FEC-Based Error Control for Internet Telephony. In *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1453 –1460 vol.3, 1999. doi: 10.1109/INFCOM.1999.752166.

A. Fecheyr-Lippens. A Review of HTTP Live Streaming, 2010. URL `http://andrewsblog.org/a-review-of-http-live-streaming`.

A. Ferragut and F. Paganini. User-Centric Network Fairness through Connection-Level Control. In *Proceedings of International Conference on Computer Communications*, pages 1–5. IEEE, 2010.

R. Kuschnig, I. Kofler, and H. Hellwagner. Improving Internet Video Streaming Performance by Parallel TCP-Based Request-Response Streams. In *Proceedings of Computer Communications and Networking Conference*, pages 1–5. IEEE, 2010.

Weiping Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):301–317, 2001. ISSN 1051-8215. doi: 10.1109/76.911157.

Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *SIGCOMM Computer Communications Review*, 27:67–82, 1997. ISSN 0146-4833.

J.R. Norris. *Markov Chains*. Cambridge University Press, 1998.

Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 303–314, New York, NY, USA, 1998. ACM. ISBN 1-58113-003-1.

R. Pantos. HTTP Live Streaming. Internet-Draft draft-pantos-http-live-streaming-05, Internet Engineering Task Force, 2010. URL `http://tools.ietf.org/html/draft-pantos-http-live-streaming-05`. Work in progress.

R. Rejaie, M. Handley, and D. Estrin. Layered Quality Adaptation for Internet Video Streaming. *IEEE Journal on Selected Areas in Communications*, 18(12):2530–2543, 2000. ISSN 0733-8716. doi: 10.1109/49.898735.

Y. Sánchez de la Fuente, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Louédec. iDASH: Improved Dynamic Adaptive Streaming over HTTP Using Scalable Video Coding. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, pages 257–264, New York, NY, USA, 2011. ISBN 978-1-4503-0518-1. doi: http://doi.acm.org/10.1145/1943552.1943586. URL `http://doi.acm.org/10.1145/1943552.1943586`.

H. Schwarz, D. Marpe, and T. Wiegand. Overview of the Scalable Video Coding Extension of the H. 264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, 2007.

Dapeng Wu, Y.T. Hou, Wenwu Zhu, Ya-Qin Zhang, and J.M. Peha. Streaming Video Over the Internet: Approaches and Directions. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):282 –300, 2001. ISSN 1051-8215. doi: 10.1109/76.911156.

A. Zambelli. IIS Smooth Streaming Technical Overview. *Microsoft Corporation*, 2009.