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

Travis Andelin
Computer Science
Brigham Young University
Provo, UT 84602
travis.andelin@gmail.com

Vasu Chetty
Computer Science
Brigham Young University
Provo, UT 84602
chettyv@byu.edu

Devon Harbaugh
Computer Science
Brigham Young University
Provo, UT 84602
dharbaugh@gmail.com

Sean Warnick
Computer Science
Brigham Young University
Provo, UT 84602
sean@cs.byu.edu

Daniel Zappala
Computer Science
Brigham Young University
Provo, UT 84602
zappala@cs.byu.edu

ABSTRACT

Video streaming on the Internet is increasingly using Dynamic Adaptive Streaming over HTTP (DASH), which allows a client to dynamically adjust its video quality by choosing the appropriate quality level for each segment based on the current download rate. In this paper we examine the impact of Scalable Video Coding (SVC) on the client's quality selection policy. 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? We use a combination of analysis, dynamic programming, and simulation to show that a client should use a diagonal quality selection policy, which combines prefetching with backfilling to balance both of these concerns. We also illustrate the conditions that affect the slope of the diagonal policy.

1. INTRODUCTION

Gaining popularity among common methods of video streaming is a generic protocol known as *Dynamic Adaptive Streaming over HTTP* (DASH) [2]. 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 [9, 13]. 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 short (e.g. two second) segments. Each segment is typically 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.

Scalable Video Coding (SVC) is an extension to a the H.264/ MPEG-4 Advanced Video Coding standard for video compression [5]. 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. [12] shows that there is a 10% increase in the bitrate for spatial and SNR scalability.

We believe the advantages offered by SVC 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 short 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 shows a possible state in downloading a video. The light gray blocks are those that have been downloaded, and the dark gray 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.

Of course, the quality selection policy will depend on the dynamics of the available bandwidth for the client. It is

*Developed with support from Move Networks. A full version of this paper that includes proofs and additional details is available as a technical report at `ilab.cs.byu.edu`.

MMSys'12 February 22-24, 2012, Chapel Hill, North Carolina, USA.

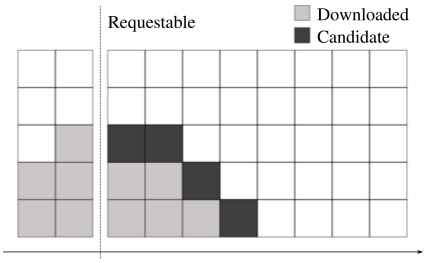

Figure 1: A DASH with SVC client must decide whether to download for the current segment or for a future segment.

well known that the bandwidth a client can obtain will vary based on load, loss, and delay [6, 8], 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 we examine here is *under which conditions should a client download with a diagonal policy, and how can it select the slope of this policy?* By a diagonal policy, we mean one where the client alternates periods of prefetching (downloading for future segments) and backfilling (downloading for the current segment). We first explore which policies we can show to be optimal, and under which conditions. We prove that variations of a vertical policy are optimal under several constant-rate scenarios. We also use dynamic programming to show that when the rate is variable and modeled with an independent, identically distributed (i.i.d.) random variable, then the optimal policy is diagonal and the slope varies based on the parameters of the rate distribution. We then use simulation to tackle larger-scale problems with more general rate models. We compare a variety of policies and show that as the variation of the rate increases, the best quality selection policy becomes diagonal. A flatter slope is needed when (1) the rate is low, (2) the rate is highly variable, and (3) the rate persistence is high and the overall rate has a chance of being low.

2. PROBLEM FORMULATION

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. We assume the size of each block of video to be constant, though in reality it will vary slightly with SVC coding. A downloaded video can be represented as a vector $q \in \{0, 1, \ldots, b\}^N$, where each element q_i represents the number of blocks downloaded for segment i out of the b blocks available. This can also be called the quality of that segment.

The video is stored on a server, and is simultaneously downloaded and played by a video client across a network. Downloading begins at time zero and continues every k seconds until time kN; playback of the first segment begins at k seconds. At each decision point, the client chooses a segment, then requests the next block for that segment. 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 path between the server and the client is characterized by a rate and delay. Let the rate behave as some stochastic process, and let w be the random variable that characterizes the rate distribution measured during each k-second interval. In discussing optimal solutions we treat this rate as blocks per segment, and in our simulations we consider it to be in terms of blocks per second. Let d represent the round-trip delay between the client and the server. As delays typically have small variance, we assume d to be constant throughout the download.

The user watching the video is happy when the video quality is smooth and maintains high quality over time, and is particularly unhappy when the playback quality for any segment is 0. Such an occurrence means that the user must either wait as the client finishes downloading segments for that block, or miss out on that k-second segment altogether.

We thus use the following metric to determine the user's happiness with the video streaming experience, after it is finished:

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

The left term is the *geometric mean* of qualities, which we refer to as the *quality score*. Considering all possible vectors of quality q for which $\sum_{i=0}^{N-1} q_i = j$, 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 quality selection policy will maximize the expected value of y?*

3. OPTIMAL POLICIES

We examine the optimal quality selection policy under several assumptions about the download rate. We assume that $d = 0$, to omit round-trip delay from consideration.

3.1 Constant Rate

We first examine the case when the download rate is steady and can be modeled as a constant r in blocks per segment. A *Vertical* policy is one that will always improve quality for the current segment if at all possible before considering the next segment. Intuitively, it is not difficult to see that a vertical policy will do well, with a constant download rate.

When formulating a vertical quality selection policy, there is a tradeoff between minimizing the buffer size and minimizing quality variation. We illustrate this tradeoff by defining two different types of vertical policies: *FloorVertical* and *MeanVertical*.

With the *FloorVertical* policy, the client downloads $\lfloor r \rfloor$ blocks in every time step, plus one extra block when enough extra bandwidth accumulates. Denote the extra bandwidth available during a segment as $\alpha = r - \lfloor r \rfloor$. 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:

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

THEOREM 1. *For a constant rate r, when quality variation need not be minimized ($\lambda = 0$), but buffer space should be minimized, then the optimal policy is FloorVertical.*

This may be useful for a mobile client, for example, with limited memory. To minimize quality variation, while still maximizing the quality score, we define the *Mean Vertical* policy, in which the client downloads all completable blocks at a quality $\lfloor r \rfloor$, for $h = \lceil N\beta \rceil$ time steps, and using excess time to download at a quality of $\lceil r \rceil$ blocks starting at time step $h+1$, where $\beta = 1 - \alpha$, such that $q_i = \lfloor r \rfloor$, for $i \leq h$, and $q_i = \lceil r \rceil$, for $i > h$.

THEOREM 2. *For a constant rate r, when the quality metric takes into account quality variation ($\lambda > 0$), then the optimal policy is MeanVertical.*

The maximum buffer size needed for the Mean Vertical policy is $\alpha \lceil N\beta \rceil$.

Proofs for the above theorems can be found in a technical report [3].

3.2 Variable Rate

We now use dynamic programming to examine the case where where the client's download rate is variable. To make the video streaming problem solvable with dynamic programming, we re-formulate the problem slightly, as follows. First, we assume no bound on b, the maximum quality for a segment. Second, download decisions are made at fixed, discrete intervals, rather than after the previous block is entirely received. 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 $N * m$ time steps. $u \in \{0,1\}^N$ is an index vector, with a 1 in the position indexing the segment to download, and with 0s in every other position. Finally, 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, and at the end of the time step, it has received a fraction of a block according to w.

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. The objective to this problem is the function $y(\lfloor x \rfloor, 0)$ from Equation (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 Section 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.

N	m	Memory (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 1: Approximate memory and time required to find solutions for various sizes of dynamic programming problems.

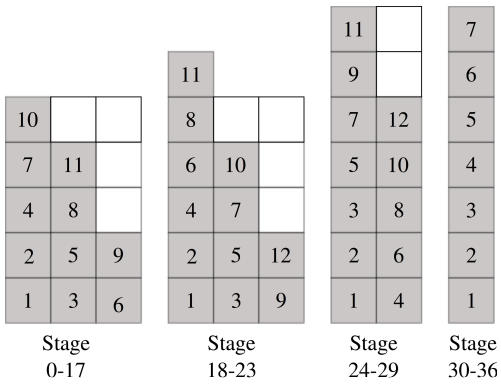

Figure 2: The optimal quality selection policy at different stages of a video, where $N = 6$ and $m = 6$. $P(w = 0) = 0.5$, $P(w = 1) = 0.5$.

The optimal policy is straightforward to compute by using dynamic programming. We implement the solver as a Java program that 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 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.

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 distribution on blocks received per m attempts, or in other words, blocks received per segment interval.

By examining the optimal solutions for some choices of w, we learn several key principles. First, the optimal policy follows a diagonal pattern, where the client alternates periods of backfilling and prefetching. Figure 2 illustrates the quality 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 of blocks showing the quality selection policy at different different stages of the video download. The client selects the blocks in the order in which they are labeled.

Second, the slope of the diagonal policy becomes steeper toward the end of the video, because there are fewer seg-

8	10			
4	6	7		
1	2	3	5	9

(a) Stage 0-11 for P(w=1) = 0.25

6	
4	
3	8
2	7
1	5

(b) Stage 0-21 for P(w=1) = 0.75

Figure 3: The optimal quality selection policy at different stages of a video, where $N = 6$, $m = 6$.

ments in the future that can benefit from prefetching. The grids to the right in Figure 2 show the same video at later stages, with steeper policy slopes due to fewer total segments left to play.

Third, we note that the slope of the diagonal policy depends on the rate distribution. Figure 3 displays the optimal policies at the beginning of the video for two different distributions. The one on the left has $P(w = 1) = .25$, so the distribution over segment intervals has a low mean that is right skewed. The optimal policy in this case is a flat diagonal, meaning more prefetching. The one on the right has $P(w = 1) = 0.75$, so the distribution has a high mean that is left skewed. The optimal policy in this case is a steep diagonal, meaning more backfilling.

4. SIMULATION STUDY

Dynamic programming gives us useful information about policies that work well with different kinds of rate distributions, however it is limited to small videos and requires a reformulation of the problem that may not be realistic. To explore how policies perform on larger problems, with more general bandwidth distributions and more realistic client behavior, we conduct a simulation study.

We model the available bandwidth to TCP with both a truncated normal distribution and a Markov chain. 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.

We compare the *Vertical*, *Mean Vertical*, and *Diagonal* quality selection policies, where the slope of the Diagonal policy varies between $-5°$ and $-85°$ in decrements of $5°$. We also include a purely *Horizontal* policy, which downloads the entire video in quality 1, then quality 2, and so on until the movie is finished. In this case, the probability of having a zero quality segment will clearly be minimized.

We create an event-driven network simulator to measure the performance of these policies under varying conditions. The simulator takes as inputs the video length N in segments, the segment duration in seconds, the network round-trip delay d, the rate model, and the quality selection policy.

4.1 Truncated Normal Rate Model

The truncated normal distribution models the download rate using μ, the mean of the normal distribution; σ, the standard deviation of the distribution; *min*, the minimum

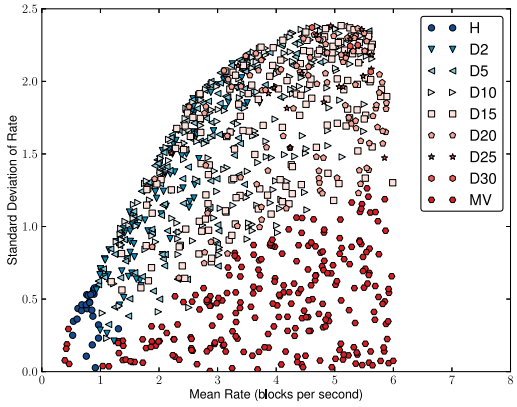

Figure 4: Best-performing policies under truncated normal rate model, with quality variation ($\lambda = 1$.)

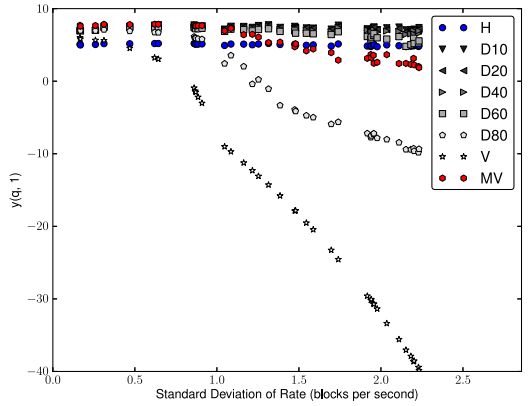

Figure 5: Performance with truncated normal rate model, $\lambda = 1$, $3.75 < mean\ rate < 4$.

outcome of the distribution; *max*, the maximum outcome of the distribution; and *interval*, the duration for each generated rate.

We run an experiment with the following methodology. We fix *min* and *max* 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 4 shows how the mean and standard deviation 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.

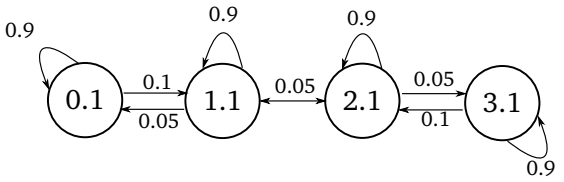

Figure 6: Example Markov chain rate model.

To see how each policy's score is affected by increasing standard deviation of the rate, we plot in Figure 5 the performance of each policy for the range where the mean download rate is between 3.75 and 4. 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. The Horizontal policy uses much more buffer space than the other policies, requiring a cache of nearly 5000 blocks, since it always downloads the minimum quality for each segment before proceeding to higher qualities. The other policies all require a cache of at most 500 blocks, and with the Diagonal policies the buffer space used declines slightly as the slope of the policy increases.

4.2 Markov Rate Model

An important aspect of performance not captured in the truncated normal 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 change is somewhat continuous, e.g., a rate is more likely to change from 5Mbps to 4Mbps than from 5Mbps to 1Mbps. We use a Markov chain [7] to model the rate so that it includes rate persistence. We let the nodes represent rates, and the arcs represent transition probabilities to adjacent nodes. See Figure 6 for an example of a Markov chain.

The Markov rate manager takes as input such a Markov chain, with a set of nodes and transition probabilities. 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 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. 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 6 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 7 shows which policies perform best for different expected rates and different remain probabilities. From this figure, it is clear that very low rates coupled with very high remain probabilities favor the Horizontal policy. Flat Diagonal policies work best when

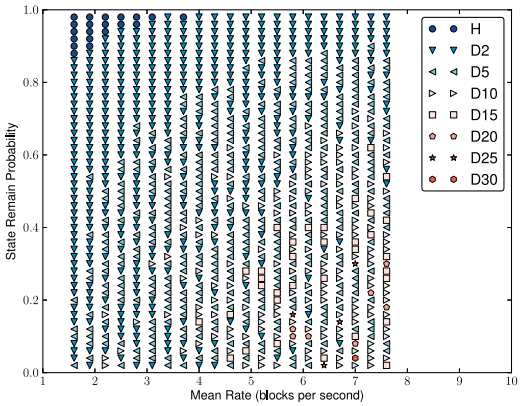

Figure 7: Best-performing policies under Markov chain model, with quality variation ($\lambda = 1$.)

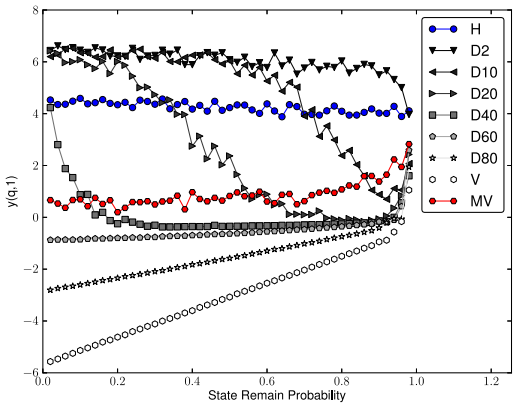

Figure 8: Performance with Markov chain model, $\lambda = 1$ and *mean rate* $= 3.75$.

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 minimal, 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, we plot in Figure 8 the performance of each policy where the mean download rate is equal to 3.75. The flattest Diagonal policy does best overall, while the other 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, the Horizontal policy by far has the largest buffer requirement, requiring a cache of nearly 5000 blocks, whereas most of the other policies are below 500 blocks. MeanVertical and Diagonal with a slope of -2° have buffer requirements that start at 1000

blocks and increase to 2000 and 1200, respectively, as the remain probability increases. Thus even a slight departure from Horizontal, with a Diagonal policy having a slope of $-10°$, greatly reduces the buffer requirements while still doing well on the quality objective.

5. RELATED WORK

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. [10]. 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. This protocol is designed to be used over UDP with a TCP-friendly rate control mechanism that reacts to congestion on shorter time scales. Our work uses the client to adjust the quality of the video on a per-segment basis, to adapt to the rate that TCP provides.

More recent research has explored the combination of SVC with DASH. Abboud et al.[1] examine the use of SVC in the context of peer-to-peer video-on-demand streaming. This paper examines the impact of different quality adaptation algorithms on performance. The client's buffer is divided into a high priority zone, consisting of segments that will be played soon, and a low priority zone, consisting of segments farther in the future. As soon as the requirements for the high priority zone are met, prefetching begins on the low priority zone. Sánchez et al. [11] 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.

Several studies have examined the performance of deployed DASH systems. Akhshabi et al. evaluate Microsoft Smooth Streaming and Netflix, along with Adobe's open source adaptive player [2]. The authors use experiments on a local network using a DummyNet router to identify issues in several implementations with stability, convergence, and fairness. Another study, by Fecheyr-Lippens, conducts an experiment comparing existing multimedia streaming systems with Apple's HTTP Live Streaming [4]. This examines client CPU load, client memory consumption, and server capacity.

6. CONCLUSION

It is clear that using a scalable codec offers enhancements not provided by traditional DASH architectures. We have shown that the Diagonal quality selection policy does a good job of balancing both the quality score and quality variation, while keeping buffer requirements low. We have also demonstrated the conditions under which the slope of the Diagonal policy should be flatter or steeper. A major area of future work is to develop a dynamic diagonal policy that adjusts the slope based on measurements of the rate distribution and persistence. Because many devices have limited memory capacity, it would be interesting to explore how performance of the various policies is affected when capacity is limited.

7. ACKNOWLEDGMENTS

We are grateful for the generous support of Move Networks and many insightful discussions with Drew Major and Paul Sherer.

8. REFERENCES

[1] 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. *ACM Multimedia Systems*, 2011.

[2] S. Akhshabi, A. Begen, and C. Dovrolis. An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP. In *ACM Multimedia Systems (MMSys)*, 2011.

[3] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala. Quality Selection for Dynamic Adaptive Streaming over HTTP with Scalable Video Coding. Technical Report 2, Internet Research Lab, Brigham Young University, 2011.

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

[5] W. Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):301 –317, March 2001.

[6] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *SIGCOMM Computer Communication Review*, 27:67–82, July 1997.

[7] J. Norris. *Markov Chains*. Cambridge Univ Press, 1998.

[8] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM*, pages 303–314, New York, NY, USA, 1998.

[9] R. Pantos. HTTP Live Streaming. Internet-Draft draft-pantos-http-live-streaming-05, Internet Engineering Task Force, Nov. 2010. Work in progress.

[10] 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, dec 2000.

[11] 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 *ACM Multimedia Systems (MMSys)*, 2011.

[12] 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.

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