QAVA: Quota Aware Video Adaptation

Jiasi Chen, Amitabha Ghosh, Josphat Magutt, and Mung Chiang
Princeton University, Princeton, New Jersey, USA
{jiasic, amitabhg, jmagutt, chiangm}@princeton.edu

ABSTRACT

Two emerging trends of Internet applications, *video traffic becoming dominant* and *usage-based pricing becoming prevalent*, are at odds with each other. Given this conflict, is there a way for users to stay within their monthly data plans (data quotas) without suffering a noticeable degradation in video quality? In this work, we develop an online video adaptation system, called Quota Aware Video Adaptation (QAVA), that manages this tradeoff by leveraging the compressibility of videos and by predicting consumer usage behavior throughout a billing cycle. We propose the QAVA architecture and develop its main modules, including Stream Selection, User Profiling, and Video Profiling. Online algorithms are designed through dynamic programming and evaluated using real video request traces. Empirical results suggest that QAVA can provide an effective solution to the dilemma of usage-based pricing of heavy video traffic.

Categories and Subject Descriptors

C.2.5 [**Local and Wide-Area Networks**]: Internet; H.5.1 [**Multimedia Information Systems**]: Video

Keywords

Video streaming, Video rate adaptation, Data quota.

1. INTRODUCTION

1.1 Motivation for Online Video Adaptation

This paper is motivated by two recent and conflicting trends in Internet applications, which may be summarized by two numbers: 70 and 10.

- 70 is the predicted percentage of mobile traffic from video alone by 2016 [1]. Together with YouTube, Netflix, Hulu, HBO Go, iPad personalized video magazine apps, and news webpages with embedded videos, video traffic is surging on both wireline and wireless Internet.
- 10 is the dollars per gigabyte (GB) charged by the two major US cellular carriers once a baseline data quota is exceeded during a monthly billing cycle. Tiered pricing, or usage-based pricing, is becoming increasingly commonplace in other countries and even for wireline broadband. For example, in May 2011, AT&T wireline U-Verse high-speed Internet began charging $10 per 50 GB beyond a baseline data quota. In Canada, the charges are even steeper, with Rogers charging $2 per GB overage fees on its high-speed Internet service [2]. And in India, Reliance charges 0.02 rupees per 10 kB overage on its 3G mobile data plans [3].

These two trends, *video traffic becoming dominant* and *usage-based pricing plans becoming prevalent*, are at odds with each other. On the one hand, videos, especially on high-resolution devices (e.g., iPhone 5, iPad, Android tablets), consume much more data than other types of traffic; for instance, 15 min of low bitrate YouTube videos per day uses 1 GB a month, and a single standard-definition movie can take up to 2 GB. On the other hand, usage-based pricing threatens the business model of delivering entertainment via the high speed Long Term Evolution (LTE). These factors can result in high overage charges by the service provider, subscription to more expensive data plans, or discontinuation of data service by disgruntled users. Given this conflict, a natural question to ask is: *Can a consumer stay within her monthly data quota without suffering a noticeable drop (distortion) in video quality?*

In today's practice, there are two main approaches to balancing the competing goals of delivering high quality video while consuming less data:

- Consumers may be warned by service providers (or by self-imposed warnings) to stop watching more videos once their usage-based charges become too high. This straight-forward "solution" can be undesirable as it could result in dissatisfied users [4].
- Content providers can take a *"one size fits all"* approach of cutting back bit rates across all video requests, for all users, and at all times. For example, the YouTube mobile app automatically chooses low quality video when the request is made over the cellular data network. Netflix implemented a similar approach in Canada in March 2011, in light of expensive usage-based charges by Canadian Internet Service Providers (ISPs), even for wireline customers.

CoNEXT'12, December 10–13, 2012, Nice, France.

In contrast, Quota Aware Video Adaptation (QAVA) in this paper exploits the dynamic range of video compressibility and users' temporal data consumption patterns. To the best of our knowledge, video adaptation with respect to each user's data quota in the context of usage-based pricing has not been systematically investigated.

1.2 Trading off Quality vs. Cost vs. Volume

Our premise is the following: *Not every video bit is needed for every consumer*, and the bit rates can be adjusted not only based on screen resolution and channel conditions but also usage patterns. QAVA can be customized to each user's demand and monthly data quota by adaptively choosing an appropriate bit rate for each video request, thereby *shaping the supply* for the user. We will show that by leveraging *video compressibility* and profiling *usage behavior*, QAVA can significantly mitigate the conflict between growing video traffic demand and usage-based pricing.

At the heart of QAVA is a *Stream Selector* (SS), which takes inputs from a *User Profiler* (UP) and a *Video Profiler* (VP), to select a particular bit rate and pre-emptively compress the more compressible videos early in the billing cycle. The VP provides information related to a video, such as its compressibility, which measures the extent to which the size of a video can be reduced without a significant distortion in quality. The UP predicts consumer usage patterns from past history and customizes the system to every user's flavor for watching certain types of videos. The SS then uses the information provided by the VP and UP to optimize QAVA for each user based on her monthly data quota.

The benefits to a QAVA-enabled user include the ability to watch more videos under a given monthly data plan without suffering a noticeable distortion in video quality, as compared to a non-QAVA user. Or, phrased differently, if a user's demand for video traffic remains the same or goes down, QAVA tries to save money for the user with a minimum impact on video quality. This 3-way tradeoff is illustrated in Figure 1. Across the three competing goals of minimizing cost, maximizing the number of videos watched, and minimizing distortion, QAVA strikes a graceful, tunable tradeoff.

1.3 Incentives of Players in QAVA Ecosystem

A natural question to our proposed approach is: *What are the incentives for different players in the ecosystem to use QAVA?* We address this from the perspective of three major players: users, ISPs, and content providers.

Users: A user has the most obvious incentive, because QAVA enables her to stop worrying about her monthly data plan and watch all the videos she wants with minimal distortion.

ISPs: An ISP has two options: it may wish to (a) reduce data traffic to lower network congestion and thereby operational and capital expenditure costs, or (b) preserve traffic to continue receiving overage charges from customers and/or usage fees from content providers. In the first case, QAVA ensures that all customers remain below their quota, which indirectly lowers the traffic rate. In the second case, the ISP can set the user's quota parameter to ensure that the user still consumes the same amount of data as a non-QAVA user, but receives better video quality.

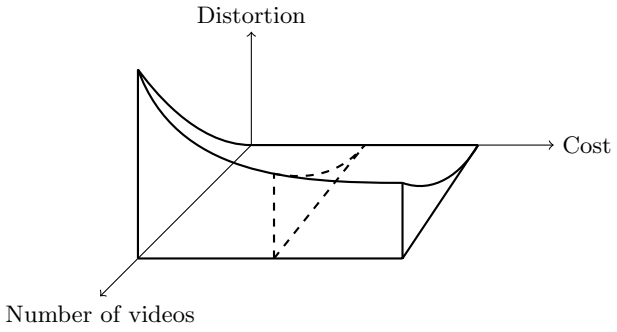


Figure 1: A 3-way tradeoff between distortion, cost, and the number of videos watched. For a fixed cost, QAVA enables a user to watch the desired videos while minimizing distortion.

Content providers: The advantages for the content provider (CP) are three-fold. Firstly, since QAVA allows a user to access more content under the same data plan, the CP achieves greater profit by increasing the content consumption and thereby advertising revenue. Secondly, the CP improves customer satisfaction by removing her worries about exceeding her quota, which can be marketed as a competitive advantage over other CPs. Thirdly, QAVA reduces the potential need for the CP to pay the ISP for the customer's data charges.

QAVA is thus mutually advantageous from the perspectives of all three players. We summarize our contributions along four different dimensions:

- **Architecture:** We design a modular system architecture for QAVA comprising three functional modules: SS, UP, and VP. Our design makes QAVA a deployable system that can be used by real-world consumers.
- **Algorithms:** We design an online bit rate selection algorithm for the SS module based on finite-horizon Markov Decision Process (MDP) [8]. This algorithm runs at the heart of QAVA, enabling it to provide a graceful, tunable control of the quality, cost, and volume tradeoff.
- **Experiments:** We evaluate the performance of QAVA in simulations using real video request traces.
- **Implementation:** We implement QAVA as an Android application and deploy it among several volunteers from the Princeton University community in an ongoing trial. Further trials are being planned with several major carriers and content providers.

The rest of the paper is organized as follows. In Section 2, we describe the QAVA system architecture and discuss design considerations. The individual modules are described in detail in Sections 3 and 4. We provide simulation results in Section 5 and implementation details in Section 6. Lastly, we discuss related works in Section 7, and conclude in Section 8.

2. QAVA SYSTEM ARCHITECTURE

In this section, we describe the architecture of QAVA and the different modules that comprise the system. For each module, we describe its functionality as well as its connection with other modules.

2.1 A Modular Architecture Design

The architecture of QAVA comprises three different modules, each responsible for a specific function. The modules work together to enable QAVA to optimize across the three performance goals shown in Figure 1. We first describe the motivation for each of the modules.

Selecting Right Bit Rates: The basic operation of QAVA is to choose an appropriate bit rate for every video request made by a user. This bit rate selection is based on two factors: (i) the user's data consumption pattern, and (ii) the particular video requested. This job is performed by a *Stream Selector* module running at the heart of QAVA on the content provider's server, as shown in Figure 2. Due to space limitation, we focus on the pre-encoded bit stream scenario, where each video has multiple copies, each copy pre-encoded in a different bit rate and stored on the content provider's server. The number of copies with different bit rates of a video is pre-determined by the content provider.

Profiling Usage Behavior: A user's past data consumption history gives an indication of her future usage pattern. Since the video requests from a user can arrive at different times without any prior knowledge, a prediction of these arrival times is helpful to QAVA so it can choose appropriate bit rates to serve the requests. A simple usage pattern could be watching on average x_d number of videos (or, equivalently, y_d bytes) on day d of a week. When the number x_d (or y_d) remains approximately the same for the same day d across different weeks, we may notice a weekly pattern. More complex usage behavior can lead to small-scale seasonal variations (daily or hourly) as well as *trends*, which are long term variations due to habitual changes that lead to a steady increase or decrease in data usage over months or years.

QAVA employs a *User Profiler* module to find patterns in usage behavior and to predict future video requests. In particular, the UP module estimates the probability of a video request arriving in a given time interval. The length of this interval can be uniform or variable depending on the past data consumption history, and is configured by a system parameter. We design the UP module as an application that can be installed on a user device (client), as shown in Figure 2.

Estimating Video Compressibility: In addition to staying within a monthly data quota, the SS algorithm also aims to minimize video distortion. For this, the algorithm needs to know to what extent the requested video can be compressed and how much distortion it would cause in doing so. Different videos have different levels of compressibility depending on their spatial or temporal activity, as well as on the choice of encoder. For example, a talk show that has very little motion in it can be greatly compressed using an H.264/AVC encoder, whereas a motion-rich sports video may not be compressible to the same extent.

The SS algorithm should be careful in choosing the right bit rate for every video request to avoid the following undesirable situation. Suppose the algorithm chooses a high bit rate for an easily compressible video when the user has

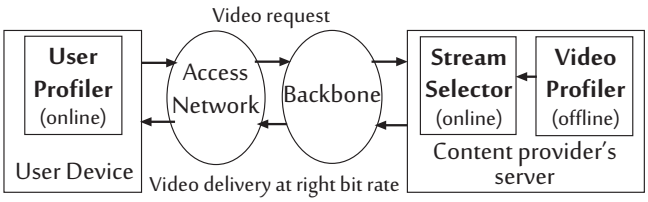


Figure 2: QAVA's modular system architecture: The UP module sits on a user device, whereas the SS and VP modules are located on a content provider's server. A video request originating from a user device travels through the access network and the backbone to the server, which then runs a stream selection algorithm to choose an appropriate bit rate to deliver to the user.

a lot of quota left, possibly in the beginning of a month. Then it might be forced to choose low bit rates for some not-so-easily compressible videos near the end of the billing cycle in order to stay within the monthly budget, thus causing significant distortion. A possible remedy is to choose low bit rates for easily compressible videos *even when* there is sufficient quota left. However, such intelligent online decisions can be made only if the system knows about the distortion vs. bit rate tradeoff for every video and can learn the quota consumption pattern over a billing cycle for each user. An example of this tradeoff is shown in a demo video at http://snipurl.com/23uozdh. QAVA employs an offline *Video Profiler* module to compute this distortion for every bit rate and store it on the content provider's server, as shown in Figure 2.

We now summarize the three modules of QAVA:

- **Stream Selector (SS):** The SS module is at the heart of QAVA and is located on the content provider's server. It is an online module which decides on the right bit rate for every video request.
- **User Profiler (UP):** The UP module predicts the probability of future video requests from past usage history, and also computes a user-specific distribution of video types reflecting the user's taste of watching different types of videos. It is an online module and is located on the user device.
- **Video Profiler (VP):** The VP module is also located on the content provider's server and is an offline entity. It computes the distortion for every bit rate version of all the stored videos. We loosely call this the "*compressibility*" of the videos.

Input-Output Connections Between Modules: Based on the preceding discussion, the relationship between the different modules describing their inputs, outputs, and update frequencies is shown in Table 1. The input to the SS module is the compressibility of the requested video, the user's remaining monthly budget, and the user's profile as output by the UP module. For every request, it runs the stream selection algorithm and outputs the selected bit rate version. The input to the VP module is the set of all videos stored on the content provider's server, and its output is the compressibility for each video. The input to the UP module

Module	Input	Output	Frequency
Stream Selector (SS)	Compressibility of video request, remaining monthly budget, and the user profile.	Selected bit rate version to deliver for the video request.	Every request
User Profiler (UP)	Time stamps and compressibility all past video requests.	Probability of a video request arriving at a time interval, and the video compressibility distribution for all past requests. Together these comprise the user profile.	Every billing cycle
Video Profiler (VP)	All videos stored in the content provider's server.	Compressibility of all videos.	Offline

Table 1: Three key functional modules of QAVA with their inputs, outputs, and update frequency.

are the time stamps for the user's past video requests, as well as the compressibility of those videos. Its output is the predicted video request probability and the video type distribution (i.e., compressibility distribution) specific to that user. These quantities characterize the user's behavior and are fed to the SS module. These predictions can be made at the beginning of the billing cycle, or updated more periodically throughout the billing cycle.

To be concrete, we give an operational example.

1. The content provider computes and stores the compressibility of all the videos on the server.

2. In the beginning of a billing cycle, the UP makes predictions (to be used in the current cycle) of the video request probability and the compressibility distribution based on its log of the past requests of the user.

3. When the user requests a video in the current billing cycle, the request is sent to the SS module, which selects the bit rate to be delivered. The content provider also sends the compressibility of the requested video to the UP. The UP logs this as well as the timestamp of the request.

4. Once the current billing cycle is over, the UP updates its predictions based on the recent request logs. Steps 2–4 repeat for the next billing cycle.

2.2 Design Considerations

There are several alternatives and variants for designing the QAVA architecture. We briefly describe these alternatives, their advantages and disadvantages, and our design decisions.

Availability of Video Versions: In this paper, we assume that the videos are pre-encoded and stored on the content provider's server. An alternative to this is on-the-fly transcoding and adaptation, which requires compressing the video dynamically at the particular bit rate determined by the SS module. This can be a time-consuming operation and has significant implementation challenges; however, it would be capable of adapting video feeds for live events. In contrast, pre-encoded streams can be selected with minimal computation, but cannot handle video streams of live events.

We also assume that the different version of the video from which the SS module chooses are supported by the channel in terms of bandwidth requirement. These sustainable video versions may be pre-selected by the CP based on typical wireless or wireline bandwidth, or chosen on-the-fly based on bandwidth estimation techniques currently proposed for use in adaptive HTTP video streaming algorithms [9].

Time Scale of Video Adaptation: There are two choices for the time scale of video adapation: (i) inter-video adaptation, and (ii) intra-video adaptation. Inter-video adaptation is choosing a single bit rate stream for entire duration of the requested video, whereas intra-video adaptation involves dividing each video into smaller clips and choosing the correct adaptation operation for each clip.

Inter-video adaptation is suitable for video clips of short duration (e.g., Youtube videos of less than 5 minutes), because the spatial and temporal activity tends to be similar throughout the duration. However, for longer videos such as movies, it is more appropriate to stream different bit rate versions for different parts of the video depending on the spatial and temporal activity. The algorithms developed in this paper apply equally to inter- or intra-video rate switching. QAVA can be used for intra-video adaptation by considering each smaller segment as a separate video request. Such intra-video switching requires synchronous bit stream switching, which can be achieved with the advent of new video streaming protocols such as MPEG-DASH [20]. QAVA can also work with existing channel-based switching algorithms by optimizing and restricting the rates available to the existing algorithm.

Heterogeneous Data Quota: Data usage under a single data plan can be decomposed into three usage layers: (i) multiple users, (ii) muliple devices per user, and (iii) multiple traffic types per device per user. QAVA's control knob is the video traffic per user per device; thus, the "video quota" per device must be set. To accomodate non-video data traffic, the video quota should be set to a percentage of the total data quota, based on historical video data usage. Running QAVA per user is also possible by aggregating video request logs across devices. This results in coarser granularity user profiling, but may also improve performance by decreasing the sensitivity to noise. For the remainder of this work, we focus on the case of a single fixed video quota and a single user with a single device, but QAVA could easily be extended to encompass the other cases as just outlined.

Module Placement: The placement of the UP and VP modules is fairly intuitive: by necessity, the VP module is located on the CP's server, since only the CP provider knows about the video characteristics. Profiling the video on the user device is not feasible due to CPU and battery limitations. The UP module logs user data, so it should be placed on the user device to alleviate user concerns over data collection and privacy.

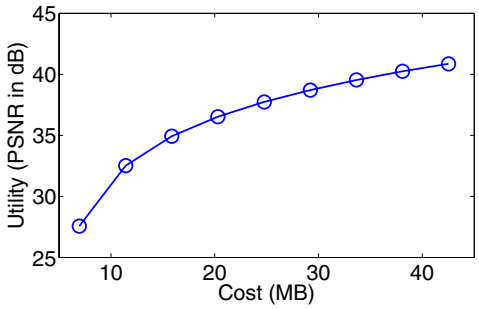


Figure 3: Video utility vs. cost showing diminishing returns for increasing cost. The x-axis represents the size of the video when encoded from 100–900 Kbps in 100 Kbps increments.

The location of the SS module that runs the bit rate selection algorithm is, however, not so intuitive. For every video request, the SS module requires inputs from both the UP and VP. One possibility is to place the SS module in the access network. Then, in order to satisfy a video request, the SS module first needs the video compressibility to be sent from the CP and the user information from the UP module. After receiving these inputs, the SS module runs the stream selector algorithm to choose the right bit rate. It then sends another request to the server to transmit the actual video in the selected bit rate. Overall, this results in unnecessary messages and potential delay, which is undesirable for delay-sensitive traffic such as video. Placing the SS module on the CP's server is thus more desirable. This also makes QAVA complementary to other video adaptation approaches [24].

Placing the SS and VP modules on the CP's server incurs some monetary cost to the content provider, which must be overcome by the advantages discussed in Section 1.3. We argue that the cost to the content provider is small: it must install the SS module on its server (one-time), and compute the video profiles of all videos (a small amount of text data compared to video data size). And regardless of where the SS module is placed, our algorithms are equally applicable.

Modularization: The QAVA system is separated into three modules: SS, UP, and VP. Each module has a fixed set of input and output and runs some internal algorithms. The advantage of this modularization is that the internal algorithms can be upgraded without changing the architecture of the rest of the system, thus simplifying testing and maintenance.

Other types of module separation and interconnections are also possible. For example, the VP and SS modules can be combined into a single module performing joint optimization. The stream selector requests videos of a certain compressibility, and the video profiler optimizes over various codecs to generate videos with desired characteristics. This would provide a finer decision granularity to the stream selector, but is computationally complex since the video profiler also runs video encoding operations.

3. STREAM SELECTION

In this section, we first describe the video request, utility, and cost model and then formulate the bit rate selection problem. We also introduce the key notation used in the paper, as summarized in Table 2.

Symbol	Meaning
u_{tj}	Utility of bit rate version j for a video request arriving at t.
c_{tj}	Cost of bit rate version j for a video request arriving at t.
x_{tj}	Indicator variable (1 if bit rate j is chosen for video request at t; 0 otherwise).
M	Number of bit rates for each video.
T	Number of time periods in a billing cycle.
$P(\mathbf{u}, \mathbf{c})$	User-specific joint probability distribution of video types based on past history.
p_t	Probability of a video request at t.

Table 2: Table of key notation.

3.1 Video Request, Utility, and Cost Model

We divide the length of a billing cycle (e.g., month) into T time intervals, indexed by $t = 1, \ldots, T$, and assume that the user has a total budget B (measured in bytes) in one billing cycle. In each time interval t, a video request arrives with a certain probability, which we denote by p_t. The remaining budget of the user at time t is b_t. The request probability p_t and budget b_t are provided by the UP module for each user.

Each video is encoded into M different bit rates, indexed by $j = 1, \ldots, M$. Associated with each video request arriving at time t is a vector of utilites $\mathbf{u}_t = (u_{t1}, \ldots, u_{tM})$, and a vector of costs $\mathbf{c}_t = (c_{t1}, \ldots, c_{tM})$ for different bit rate versions of the video.[1] When there is no video request arrival at time t, the vectors $\mathbf{u}_t$ and $\mathbf{c}_t$ are null vectors with all components being zero, because then no bit rate is selected. The VP module provides the utility and cost of each video, $\mathbf{u}_t$ and $\mathbf{c}_t$, respectively.

Each user might prefer different types of videos with different compressibilities. For example, she might want to watch news clips that have different compressibilities than sports videos. To capture this effect, we introduce a joint probability distribution $P(\mathbf{u}, \mathbf{c})$, which is user-specific and represents the probability that a user requests videos with certain utility–cost characteristics. The distribution $P(\mathbf{u}, \mathbf{c})$ is provided by the UP module for each user.

In Figure 3, we show a typical utility vs. cost function for a video encoded using the H.264/AVC codec with a resolution of 720 × 480 pixels. Such utility–cost curves are usually concave with diminishing returns for utility at higher cost (or equivalently, higher bit rate, since bit rate is proportional to data size for a fixed-length video). A video with a flat utility-cost curve is "easily compressible" because lowering the bit rate decreases the utility only slightly. In contrast, a "hard-to-compress" video has a steep curve. We measure the utility u_{tj} of bit rate version j as its *peak signal-to-noise ratio* (PSNR) × the duration of the video [6]. The cost c_{ij} is the size in bytes. Discussion of the utility and cost metrics is reserved for Section 4.2.

[1]We note that these utility and cost vectors are fixed and not time-dependent. The use of the time index t in u_{tj} and c_{tj} is purely for the ease of exposition.

3.2 Stream Selection as Knapsack Problems

We now formulate the problem of choosing the right bit rate by the SS module as different versions of the well-known knapsack problem [7] studied in combinatorial optimization. We first present an offline formulation, which is easy to understand, and then motivate the need for an online stochastic formulation.

3.2.1 Offline Multiple-Choice Knapsack Problem

The goal of the stream selector is to choose the right bit rate for every video request made by the user in a single billing cycle. We aim to maximize the sum of the utilities across all the video requests without exceeding the user's quota. In other words, we maximize the average video utility. An alternative formulation is to maximize the minimum utility across all videos requested during the billing cycle. Since the high-level goal of QAVA is to maximize the overall user satisfaction, we optimize for the average utility over time instead of the worst-case experience, as in the alternative formulation.

For a video request arriving at time t, we define a decision vector $\mathbf{x}_t = (x_{t1}, \ldots, x_{tM})$, where each x_{tj} takes the value 1 if bit rate version j is chosen, and 0 otherwise. Then our problem is:

$$\begin{aligned} \text{maximize} \quad & \sum_{t=1}^{T} \sum_{j=1}^{M} u_{tj} x_{tj} \\ \text{subject to} \quad & \sum_{t=1}^{T} \sum_{j=1}^{M} c_{tj} x_{tj} \leq B \\ & \sum_{j=1}^{M} x_{tj} = 1, \ \forall\, t \\ \text{variables} \quad & x_{tj} \in \{0,1\}, \ \forall\, t, j, \end{aligned} \tag{1}$$

where the first constraint says that the cost of the selected bit rates for all the videos requested in a billing cycle must not exceed the quota B, and the second constraint says that one bit rate version may be selected for each video.

This optimization problem is known as the Multiple-Choice Knapsack Problem (MCKP) [7]. In the regular single-choice knapsack problem, we are given a set of items, each with an associated value and weight. The objective is to pick a subset of the items such that the total value is maximized while not exceeding the knapsack capacity. In our stream selection problem, the items are the individual bit rate versions of the videos, and the multiple choices arise because exactly one version of each video must be selected.

The traditional offline version of the MCKP, where all the input items are known in advance, is well-studied. The problem is NP-hard, but pseudo-polynomial time dynamic programming (DP) solutions exist [7]. Contrary to this offline version, the SS module does not know the video requests in advance, and so needs to make decisions in an online fashion. This requires a modification to the formulation to handle online requests.

3.2.2 Online Stochastic Knapsack Problem

Unlike the traditional offline MCKP, in our scenario, the video requests are revealed one-by-one *online*. Thus, existing DP solutions to the offline knapsack problem cannot be used. Online algorithms handle this situation by making a decision on-the-fly when a new video request arrives, without any prior knowledge of future requests. However, once a decision is made, it cannot be revoked or changed in the future.

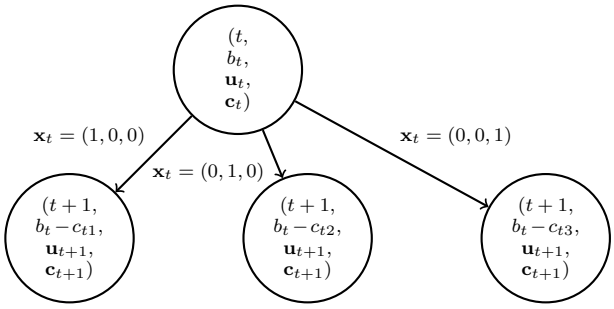


Figure 4: Stream selection modeled as a finite-horizon Markov decision process. A one step state transition is shown with 3 bit rate choices.

Finite-Horizon Markov Decision Process: Since the data quota resets after the billing cycle is over, there is a time deadline before which all actions must be made. We also note that the bit rate decisions for future intervals should not depend on the decisions taken at previous intervals, given the current remaining budget. This implies the Markov property. The problem can naturally be modeled as a finite-horizon Markov decision process (MDP). A key assumption is that the video requests are independent of time, and therefore the transition probabilities are stationary.

The MDP formulation allows the SS module to make foresighted bit rate selection decisions by taking into account the future impact of its current decisions on the long-term utility. This is better than just an online algorithm which makes myopic decisions at every time step. For example, a greedy solution might choose a bit rate that maximizes the utility of the current request without overshooting the quota.

Figure 4 shows a simple example of choosing between three different bit rates over one time step. The state of the system is defined as the four-tuple $s_t = (t, b_t, \mathbf{u}_t, \mathbf{c}_t)$, comprising the current time interval t, the remaining quota b_t, and the utility and cost vectors $\mathbf{u}_t$ and $\mathbf{c}_t$. There are three possible actions: (i) choose the lowest bit rate, i.e., $\mathbf{x}_t = (1,0,0)$; (ii) choose the second bit rate, i.e., $\mathbf{x}_t = (0,1,0)$; or (iii) choose the third bit rate, i.e., $\mathbf{x}_t = (0,0,1)$. If the lowest bit rate is chosen, the system moves to time $t+1$ with remaining budget $b_t - c_{t1}$. The algorithm collects utility (reward) u_{t1} and receives the new video request with utility and cost vectors $\mathbf{u}_{t+1}$ and $\mathbf{c}_{t+1}$. If the second bit rate is chosen, the system moves to time $t+1$, but now subtracts the cost c_{t2} from its remaining budget, leaving it with $b_t - c_{t2}$. It also collects utility u_{t2}. A similar state transition results from choosing the third bit rate.

The set of actions $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ taken by the algorithm at every time step is called a *policy*. A policy that solves the MCKP of (1) is called an optimal policy. If the arriving video requests were known, an optimal policy can be determined using the traditional offline techniques previously mentioned. However, since the video requests are not known *a priori*, the MDP finds a policy that instead maximizes the *expected* sum utility. We develop a solution using DP and online optimization.

3.3 Solving Finite-Horizon MDP

Online Optimization: The optimal policy can be found using standard backward induction techniques for finite-horizon MDPs [8]. We first define $U_t(b_t)$ as the expected utility accumulated from time t until the end of the billing cycle at time T, when the remaining quota is b_t. This expected utility assumes that the optimal action is applied in each state. Then, the optimal action at each time step t is found by solving:

$$\begin{aligned} \text{maximize} \quad & u_{tj}x_{tj} + U_{t+1}(b_t - c_{tj}x_{tj}) \\ \text{subject to} \quad & c_{tj}x_{tj} \leq b_t \\ & \sum_{j=1}^{M} x_{tj} = 1 \\ \text{variables} \quad & x_{tj} \in \{0,1\},\ \forall j, \end{aligned} \tag{2}$$

where the first constraint ensures that the cost of the selected bit rate is less than the remaining quota. The objective function has an intuitive meaning: It maximizes the current utility plus the sum of the expected utilities, subject to the remaining quota. The problem can be solved in $O(M)$ time by discarding the bit rates that violate the constraints, and then picking the bit rate j^* that maximizes the objective function. It is solved every time a video is requested.

DP Computation: Solving (2) requires the computation of $U_t(b_t)$. Since $U(\cdot)$ is an expectation over all future requests, it does not change with every new request, and thus can be pre-computed using DP before running the online algorithm.

Suppose we are at time t with remaining budget b_t, and a new video request arrives. Assuming that the algorithm chooses an optimal bit rate j^* by solving (2), the expected accumulated utility is equal to the utility of the current request, plus the future utility accumulated from time $t+1$ onward, given that we have already spent c_{tj^*} of our budget. This utility from time $t+1$ onward is unknown because the future video requests are unknown, and so we must take the expectation. Mathematically, this translates to:

$$\begin{aligned} U_t(b_t) = p_t \left(u_{tj^*} + E_{(\mathbf{u},\mathbf{c})}\left[U_{t+1}\left(b_t - c_{tj^*}\right)\right]\right) \\ + \left(1 - p_t\right) U_{t+1}(b_t). \end{aligned} \tag{3}$$

The $(1-p_t)$ term represents the probability that no video request arrives, and so no bit rate decision is made. The accumulated utility at time t is then equal to the utility at time $t+1$.

A crucial component of any DP solution is the boundary condition that allows the initial values of $U(\cdot)$ to be calculated. Our boundary condition is that the expected accumulated utility is 0 when the billing cycle is over, or the remaining budget is less than 0. The optimal action at time $T-1$ is to accept any video that fits in the remaining budget, and thus $U_{T-1}(b_{T-1})$ is known. Then using (3), the remaining entries of $U(\cdot)$ can be calculated. In this work, we choose the budget granularity to be 1, so b_t takes on possible values $1, \ldots, B$. The running time of computing the $U(\cdot)$ matrix is $O(TBM\Gamma)$, where Γ is the cardinality of the set $\{(\mathbf{u},\mathbf{c})\}$ in the video type distribution.

The online and offline components of the MDP stream selection algorithm are summarized in Algorithm 1. In the special case of two bit rates ($M=2$), our algorithm reduces to that of Papastavrou *et al.* [10]. With accurate user and video profiling, Algorithm 1 maximizes the sum utility while staying under the quota. In the case of inaccurate inputs, however, the algorithm may exceed the quota. In that case, the algorithm should simply choose the lowest bit rate, although this case never occurs in our numerical simulations.

Algorithm 1 MDP Stream Selection Algorithm

DP Computation of Utility Matrix

Input: Video type distribution $P(\mathbf{u},\mathbf{c})$, quota B, and billing cycle length T.

Output: A matrix $\mathbf{U}$ of size $T \times B$.

1. Compute each entry $U_t(b_t)$ of $\mathbf{U}$, using (3).

Online Bit Rate Selection

Input: Utility and cost vectors $\mathbf{u}_t$ and $\mathbf{c}_t$, remaining capacity b_t, billing cycle length T, and matrix $\mathbf{U}$.

Output: Optimal bit rate j^*.

1. Discard the bit rates with cost greater than b_t.
2. For each of the remaining bit rates, compute j^* that maximizes the objective function of (2).

4. USER AND VIDEO PROFILERS

In this section, we detail the functionality of the UP and VP modules. We first describe different patterns in user behavior and tastes, and then propose the user profiler algorithm. The VP module is also briefly explained.

4.1 Profiling User Behavior

The user profiler runs on the client device and characterizes each user through (i) the video request probability at each time interval, and (ii) the distribution of video types preferred by the user.

User Viewing Pattern and Taste: Depending on their lifestyles, different users have different time preferences for watching videos. For example, some users prefer watching videos on weekends rather than on weekdays, while others watch more in the evening after working hours than in the mornings. The taste in content of the users can also be different. For example, some users watch sports videos more often than movie trailers, while some others watch more news clips than music videos. Such preferences in user behavior can lead to well-defined patterns, both in terms of the viewing times and the types of the videos being watched.

The job of the user profiler is to estimate these temporal viewing patterns and video type preferences for each user. The UP module does this based on the user's past video request records, spanning either the previous billing cycle, or the entire history. In this work, we consider requests from the last billing cycle.

Computing Video Request Probability: In each time interval t, there is a certain probability p_t that the user requests a video. This request probability can either vary with each interval or be constant. As a first attempt, we compute the *average* request probability per interval, and set p_t for each interval equal to this average. The average request probability is computed by summing the number of requests in the previous billing cycle and dividing by the number of periods T. The time interval should be set small

Algorithm 2 User Profiling Algorithm

Input: Time stamps and utility–cost vectors $(\mathbf{u}_t, \mathbf{c}_t)$ of each video request in the previous billing cycle.

Output: Video request probability p_t, $\forall t$, and the video type joint probability distribution $P(\mathbf{u}, \mathbf{c})$.

1. Count the number of requests n_r, and the number of time intervals T in the previous billing cycle.
2. Compute average request probability as $\bar{p} = n_r/T$, and set $p_t = \bar{p}$, $\forall\, t$.
3. Count the number of times each $(\mathbf{u}_t, \mathbf{c}_t)$ pair appears in the past; denote this count by $n_{(\mathbf{u}_t, \mathbf{c}_t)}$.
4. Construct the joint probability distribution by computing the individual probabilities as: $p(\mathbf{u}_t, \mathbf{c}_t) = n_{(\mathbf{u}_t, \mathbf{c}_t)} / \sum_{(\mathbf{u}'_t, \mathbf{c}'_t)} n_{(\mathbf{u}'_t, \mathbf{c}'_t)}$

enough so that the average request probability is less than 1, but not so small as to inhibit the computation of (3).

There are several alternative approaches, including fitting distributions and prediction-based techniques. The arrival rate of videos might follow a particular known distribution (e.g., Poisson), in which case the probability of an arrival can be computed directly from the distribution itself. Alternatively, one can use more sophisticated time series analysis techniques. For example, at the beginning of the billing cycle, one can predict the sequence of future viewing times in the upcoming billing cycle, then compute the average request probability by adding up the predicted number of requests, and finally dividing that by the number of intervals. One can also design online algorithms, such as predicting the sequence of viewing times for intervals $t+1, t+2, \ldots, T$, while at interval t, and updating the predictions when a new video request arrives. Such alternatives trade off accuracy versus computation need.

We have developed one such online algorithm based on "triple exponential smoothing" [11]. However, here we will employ the simple averaging technique previously mentioned. The resulting computation requires less memory and power, and can be performed easily on a resource-constrained (in terms of battery and memory) client device. Our goal is not necessarily to develop the best user profiler, but to find a method that works well in the system as a whole. To establish this, we run trace-drive simulations in Section 5 to compare the performance of QAVA when the UP module uses the average request probability, to a scenario when the UP module has perfect knowledge of future arrivals. We find that our technique, while simple, achieves close to optimal performance (more than 95% on average).

Computing Video Type Distribution: The joint probability distribution $P(\mathbf{u}, \mathbf{c})$ reflects a user's preference for watching different types of videos. For example, a user who watches a lot of sports videos (which are not-so-compressible) will have a different distribution from a user who watches a lot of talk shows (more compressible). This video type distribution can remain the same over the length of a billing cycle, or can be time-dependent, reflecting, for instance, the fact that a user watches more sports videos at night and more news clips in the morning. As a first-order approximation, we assume that the distribution does not change with time. The distribution is computed once at the beginning of a billing cycle based on the video requests in the last billing cycle.

Our method is as follows: Each video request arriving at time interval t in the previous billing cycle has a $(\mathbf{u}_t, \mathbf{c}_t)$ pair associated with it. The probability distribution is calculated by counting the frequency of each $(\mathbf{u}_t, \mathbf{c}_t)$ pair from the last billing cycle, and then normalizing appropriately to form a probability distribution. Since the utility and cost are continuous variables, they can be binned for greater computational efficiency; however, in our dataset we find this optimization unnecessary. Through simulation, we show that this estimate performs very well, compared to the ideal scenario when the distribution of the requested videos is perfectly known ahead of time. Our user profiling algorithms are summarized in Algorithm 2.

4.2 Profiling Video Cost and Utility

The purpose of the video profiler running on the VP module is to estimate the utility and cost for all the bit rate versions of all videos stored on the content provider's server. There are many estimation techniques for computing the quality of a video. One standard objective metric is the PSNR, which we employ. The PSNR is a well-known objective metric for measuring video quality. A video typically comprises a sequence of images, and the PSNR for each image is defined as a function of the mean square error (MSE) between the original and compressed image. Mathematically, it is expressed in the logarithmic unit of decibel (dB) as $\text{PSNR} = 10\log_{10}(Q^2/D)$, where D is the pixel-wise MSE between the original and reconstructed image, and Q is the maximum pixel value (usually 255). We compute the video PSNR as the PSNR averaged over all images in the sequence. Typical PSNR values for lossy video compression are between 20 and 30 dB, where higher is better. To account for the duration of the video, we set the utility equal to the PSNR $\times$ the video duration. An example utility-cost curve is shown in Figure 3.

There exist other, potentially more accurate, metrics of video utility (e.g., Mean Opinion Scores [12] or MOS), as well as means of calculating subjective metrics from objective measurements [13, 23, 26]. However, we choose PSNR as it can be easily computed using a mathematical formula (in constrast to MOS that requires time consuming human experiments) and is very well known to the multimedia community.

Measuring the cost of a video in bytes naturally follows from the fact that the data quota is measured in bytes. The video profiler calculates the utility and the cost in MB for all the videos only once. These utility and cost vectors are stored alongside the videos on the content provider's server.

5. PERFORMANCE EVALUATION

We evaluate the performance of QAVA stream selection by comparing it with three alternatives: (i) a hindsight-based offline, optimal algorithm that knows all the video requests in a billing cycle ahead of time; (ii) a worst-case online algorithm; and (iii) a naïve method (used by, for example, Netflix). We also explore the sensitivity of QAVA to user profiler prediction errors.

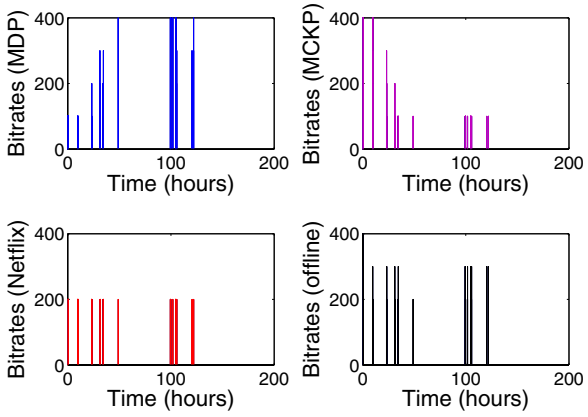


Figure 5: Bit rate selection by different algorithms for a single user. The MDP and MCKP algorithms choose different bit rates over time, while Netflix chooses a constant bit rate and the offline algorithm chooses the optimal bit rates.

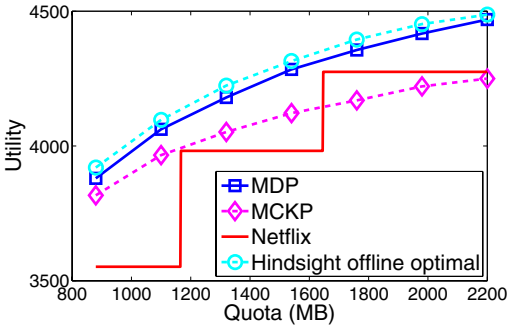


Figure 6: Quality–cost tradeoff for a single user with different quota and fixed video requests. MDP obtains close to optimal utility, while MCKP and Netflix perform sub-optimally.

5.1 Experimental Setup

Our simulations are based on the public-domain traces of 2 weeks of YouTube video requests from a wireless campus network [14]. The data comprises 16,337 users making a total of 611,968 requests over 14 days. YouTube is the largest source of video traffic, so the dataset captures a major portion of video viewing behavior [25]. The first week of trace data is used to train both video and user profilers. The second week emulates a billing cycle where the stream selector is run for each user's requests.

Each video is encoded in H.264/AVC at 100, 200, 300, 400, and 500 Kbps. The stream selector chooses one bitrate from the first four choices. The 500 Kbps version is treated as a reference for computing the PSNR of the other bit rates. The cost of each video is its size in MB. We set the user's quota at the halfway point between the minimum data usage (always selecting 100 Kbps) and the maximum data usage (always selecting 400 Kbps), and also sweep across quotas when appropriate. The period length is set to 30 minutes, since we experimentally find that varying the period length does not greatly change system performance.

One limitation of this evaluation is that not all videos were available from YouTube at the time of this study. In the training phase, missing videos are not included while generating the video type probability distribution. In the test phase, the utility–cost curves of missing videos are sampled from the video type distribution of the training phase. This gives an advantage to our algorithm, because the probability distribution of the training phase is similar to that of the test phase. We also examine the effect of misestimation of the distribution.

5.2 Comparing Stream Selection Algorithms

We first evaluate the offline algorithm that solves problem (1) with the knowledge of all future video requests. It achieves the best possible performance and is treated as the benchmark against which we compare the performance of the online algorithms. We call this the *hindsight offline optimal* algorithm.

Zhou *et al.* present an online algorithm to solve (1) with a worst-case performance guarantee regardless of the sequence of video requests [15]. We call this the online *MCKP* algorithm and chose this to compare with our MDP algorithm because it optimizes for the *worst-case* performance, while our algorithm optimizes for *expected* performance. The MCKP algorithm, however, uses less information than our MDP algorithm, needing only the maximum and minimum utility-to-cost ratio across all requested videos, and an estimate of the sum data of the smallest bit rates. The MCKP algorithm does not use prediction or time deadlines, but requires only the quota.

The second online algorithm we compare with is the naïve solution currently used by Netflix. Netflix allows subscribers to select a default streaming bit rate. We assume that a Netflix user chooses one bit rate for the entire billing cycle, and is also intelligent enough to presciently choose the maximum bit rate that fits all videos in the quota. *Clearly, this algorithm is an ideal algorithm and not suitable for practical use, because it assumes advance knowledge of the number of videos to be watched.* Comparison with this Netflix method allows us to evaluate how our MDP algorithm performs against an existing practical solution.

5.3 Single User Examples

To understand the operation of the stream selector, we first run the algorithm for a single user with a target quota of 1426 MB (see Figure 5). The video requests arrive in bursts, and each algorithm selects bit rates. The Netflix method always chooses 200 Kbps. The MDP algorithm has foresight and thus chooses lower bit rates in the beginning of the billing cycle, knowing to save for later. The MCKP algorithm does not use time deadlines, only the remaining quota, and so it chooses high bit rates in the beginning before cutting back as it starts depleting the quota. The offline optimal algorithm chooses a variety of bitrates over time.

We also sweep across different monthly quotas and measure the utility obtained by each algorithm (see Figure 6). The offline algorithm performs the best, with MDP and MCKP close behind. The Netflix method exhibits a staircase-like shape because as the quota increases, the default user-selected bit rate can increase. In all cases, the algorithms use less data than the target quota.

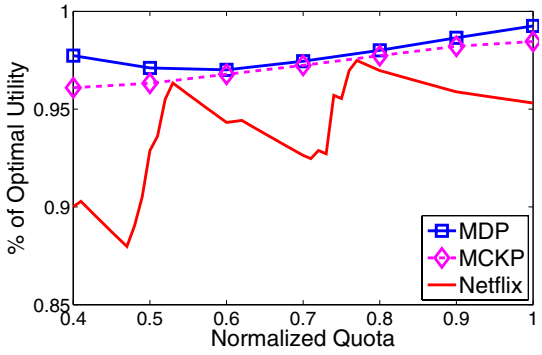


Figure 7: Quality–cost tradeoff averaged and normalized over multiple users. The MDP algorithm achieves nearly optimal performance, with the MCKP algorithm close behind. Both algorithms outperform the naïve Netflix method.

5.4 Multi-User Stream Selection

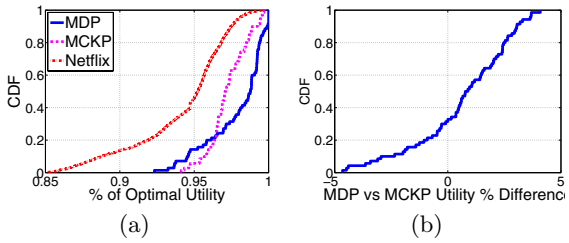
Average Performance: We now present the evaluation results of the MDP algorithm for multiple users. Each trial takes as input a fixed set of video requests and a quota, and computes the utility obtained by each algorithm. Some input combinations achieve higher utility than others. In order to fairly compare across multiple trials, we normalize the utility across different users by measuring the utility of the online algorithm as a fraction of the offline optimal utility. To normalize the quota, we measure data as a fraction of the total data if the 400 Kbps bit rate were always selected.

Figure 7 shows the average utility across 10 different users. We observe that, on average, the MDP algorithm performs better than the MCKP algorithm. The Netflix method resembles a staircase function as in the single-user case, and obtains especially low utility for low quotas. This is arguably the most important scenario: When the user's quota is small compared to the number of videos she wishes to watch. For these low quotas, the MDP has a definite advantage over MCKP, which in turn outperforms the Netflix method.

Performance Variability: To examine the utility distribution, and not just the average, we plot their cumulative distribution functions (CDFs) in Figure 8(a) across multiple quotas and users. The ideal result is a step function at 1, indicating 100% of the trials result in optimal utility. We see the Netflix method performs the worst, obtaining, for example, less than 95% utility 50% of the time. The MCKP curve is steeper, indicating it has less performance variation, which makes intuitive sense as the algorithm optimizes for the worst-case. The MDP algorithm optimizes for the average-case performance but not the spread, and thus exhibits greater variation, but is closer to the ideal step function.

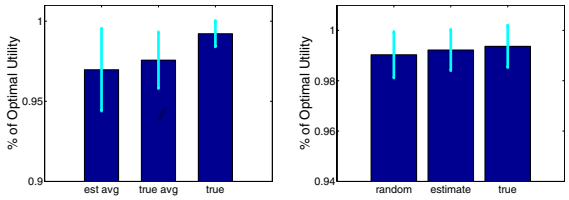
The MDP and MCKP algorithms are further compared in Figure 8(b), which shows the CDF of their percentage utility difference. If the MDP were always better than the MCKP, we would see a step function at 0. However, we observe that the MCKP sometimes outperforms the MDP algorithm.

On the surface, these simulations seem to suggest the Netflix method performs reasonably well in general. It achieves above 85% of the optimal utility, suggesting that this naïve solution is acceptable for most users. However, the caveat is



(a) (b)

Figure 8: (a) CDF of the utility achieved by MDP, MCKP, and Netflix algorithms. (b) Distribution of MDP performance improvement over MCKP.



(a) Arrival rate (b) Video type

Figure 9: Effect of user profiler prediction error.

that our simulated Netflix method assumes perfect knowledge of the number of video requests in the billing cycle, so that the user knows how to correctly set the default bit rate given the quota. This is represented by the sharp jumps with increasing quota in Figure 6 and 7. If the user sets the default bit rate too high, she will overshoot her quota. If the user sets the default bit rate too low, she will obtain suboptimal utility. A main advantage of QAVA is that it automatically adjusts the bit rate, so the Netflix user does not need to estimate her usage to set her default bit rate, which might result in these over- or under-shooting problems.

5.5 Sensitivity to Prediction Error

It is important to examine how errors from the UP module affect the performance of the stream selection algorithm. There are two possible sources of error: video request probability and video type distribution. To test sensitivity to request probability error, we measure the utility obtained by the MDP algorithm when it uses (1) the estimated average arrival rate trained on historical data, (2) the true average arrival rate of the test data, and (3) the true request times. These results are averaged across multiple users and shown in Figure 9(a), with the error bars indicating standard deviation. We observe that the greater the information accuracy, the greater the average utility. The performance difference is quite small, which suggests that the MDP algorithm performs well independent of arrival probability accuracy. Moreover, the average percentage difference between the true and estimated arrival rate is 8%.

To analyze the sensitivity of the stream selector to video type prediction errors, we perform the following experiment. We calculate the true video type distribution of the test data, and also randomly generate a video type distribution. This random distribution has both random videos (drawn from the pool of requested videos from all users), and random probabilities. The utility obtained by the MDP algorithm using the random, estimated, and true distributions,

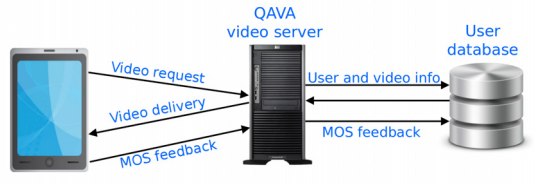


Figure 10: Android implementation of QAVA for running real-world trials within the Princeton University community.

averaged across all users, is shown in Figure 9(b) with the error bars indicating standard deviation. We find that the average utility increases only slightly as more accurate information is known, suggesting robustness of the MDP algorithm to video type distribution errors.

6. DISCUSSION ON IMPLEMENTATION

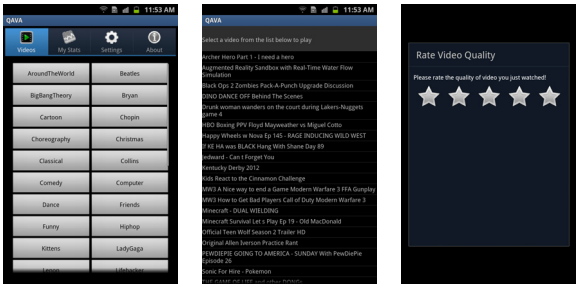
6.1 Implementation on Android

In addition to evaluating QAVA through simulations with YouTube video request traces, we have implemented QAVA to run on the Android platform. This enables us to evaluate the performance of QAVA on a commercial operating system using real users' video requests. We are currently conducting a trial, starting with 15 community volunteers.

In this section, we briefly summarize the implementation of QAVA, as shown in Figure 10. Approximately 500 videos are stored in our server, each with a resolution of 640×480 pixels and encoded using H.264/AVC in 5 different bitrates: 100, 200, 300, 400, and 500 Kbps. Participants can install the QAVA application on their mobile devices and request any of the stored videos. The videos are streamed from our server at a bit rate determined by the SS module on the server. The server also logs the time stamp, user ID, video name, and video utility and cost in a database. When the user stops watching the video, she is asked to provide a feedback on the video quality by choosing a rating on a scale of 1–5, where 1 indicates "very annoying", and 5 indicates "imperceptible distortion".

Screenshots of the GUI are shown in Figure 11. When the users opens the QAVA application, she is presented with the screen in Figure 11(a), which lists different video categories, such as music, movies, cartoons, etc. Each category contains a list of videos, which is shown on the second screen in Figure 11(b) once the user selects a category. By selecting any of the videos listed under that category, the user can start watching the video. The last screen in Figure 11(c) shows the feedback mechanism, where the user rates the video by selecting one of the 5 stars. A video demonstrating our GUI can be found at `http://snipurl.com/23upjft`.

To increase the choice of content available to trial participants, we have also used the YouTube API to implement a feature that allows users to access any YouTube video through the QAVA application. We first store a large number of YouTube videos on our server. If the user requests a YouTube video that was already pre-fetched, we run QAVA; otherwise, the video is streamed from YouTube.



(a) (b) (c)

Figure 11: Screenshots of QAVA Android app. (a) Different video categories; (b) List of videos within a category; and (c) Feedback screen.

6.2 Client-based Architectures

Our trial architecture of Figure 10 is sufficient when the SS module is located on the CP and can choose which bit rate video to stream to the user. In a practical scenario, it may be that the bit rate selected by the SS module cannot be streamed by the CP, either due to unavailability of video versions, or lack of support from the CP. To address this, we are exploring options for a *client-based* implementation architecture for QAVA. There are two possibilities: a transcoding-based system, and a throttling-based system.

In the transcoding-based solution, all traffic from the client is routed through a web proxy. By inspecting the packets, video traffic may be distinguished and transcoded on-the-fly by the proxy server to the correct bit rate. This approach has the advantage of using standard web proxy technology, and can handle all types of videos as long as the transcoder has the appropriate codec. However, there are significant implementation challenges in terms of latency minimization: the proxy and transcoder must perform quickly in order to satisfy the demands of delay-sensitive video traffic. Moreover, implementing a transcoder for HTTP video streams is also difficult.

The throttling-based solution leverages the emerging popularity of adaptive HTTP video streaming [20]. In this technology, the videos automatically adjust their bit rate based on their estimate of TCP throughput. By limiting the bandwidth observed by the rate switching algorithm, QAVA can force the video bit rate to automatically settle to the rate determined by the SS module. This throttling can be performed on the client device or on a web proxy. In terms of implementation, this approach is simpler than the transcoding-based solution, but it suffers from the limitation that only adaptive HTTP video streams can be modified in this way.

7. RELATED WORK

Adapting video quality with respect to resource constraints and user utility has been extensively studied since the 1990s, as surveyed for example by Chang and Vetra [16]. The algorithms and systems to perform such video adaptation have long been studied in the research community [9, 18, 19, 24]. There has also been increasing interest from industry: Apple, Microsoft, and Adobe have developed proprietary HTTP video streaming protocols that perform intra-video bit rate switching to adapt to varying channel conditions.

Recent progress by an industrial consortium has resulted in the MPEG-DASH standard [20], which aims to address the lack of inter-operability between current video streaming protocols by providing an open interface to access the quality levels of a video. However, these lines of work still focus on channel-based video adaptation. To the best of our knowledge, video adaptation with respect to each user's *data quota* in the context of usage-based pricing has not been systematically investigated prior to this work.

Jan *et al.* [21] developed a system for compressing images on webpages under a data quota. Although the motivation is similar, the application to images and webpages is different. It also differs in that the decisions are made with full knowledge of the images to be compressed. Compared to this, QAVA needs to make *online* decisions without the knowledge of future video requests.

Recently, systems like Onavo [22] enable users to save on data plans by forwarding all data through a proxy server and compressing images and text. However, they do not exploit consumer usage patterns or the dynamic range of video compressibility. To this end, QAVA offers substantial data saving opportunities, because video traffic is much more compressible (than images and text), and comprises the bulk of mobile data consumption.

8. CONCLUSIONS

The continuous rise of mobile video traffic and the $10 per GB usage-based pricing are at odds. Managing demand through Smart Data-Pricing (SDP) is a possible solution approach, e.g., time-shifting delay-tolerant traffic [17]. Adapting video quality offers another approach, especially for streaming video traffic. We presented QAVA, a system to mediate the conflicting trends of increased video streaming on mobile devices vs. usage-based pricing by cellular and broadband ISPs. QAVA automatically selects the best bit rate to enable the consumer to stay under her data quota, while suffering minimal distortion. By evaluting the performance on real video traces, we demonstrated that QAVA performed better than existing approaches in literature and practical solutions.

Our major ongoing work is the development and testing of user trials, in collaboration with several carriers and content providers in the US, as well as improvement of the UP and VP algorithms. A future area of exploration is the longer timescale problem of video bit rate selection and placement on limited capacity servers. Another interesting issue is the feedback between user behavior and QAVA bit rate delivery, which we have ignored so far, but merits further study. Other possible extensions include more sophisticated user and video profiler methods; bit rate selection with respect to heterogeneous devices and quota; and integration of the QAVA algorithms with upcoming video streaming standards.

Acknowledgements

This work was supported by AFOSR FA9550-09-1-0643, NSF CNS-1011962, and PECASE N00014-09-1-0449. We would like to thank our colleagues, Sangtae Ha, our shepherd Oliver Spatscheck, and the anonymous reviewers for their valuable comments.

9. REFERENCES

[1] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update", 2011-2016.

[2] "Rogers Hi-speed Internet FAQ", <http://www.keepingpace.ca/faq.html>

[3] "Reliance 3G Plans & Pricing", <http://www.rcom.co.in/Rcom/personal/3G/HTML/PostpaidDataPlans.html>.

[4] *DataWiz*, <http://www.datami.com/>.

[5] Troianovski A, "AT&T May Try Billing App Makers", *Wall Street Journal*, Feb. 28 2012.

[6] Bovik A, *The Essential Guide to Video Processing*, Elsevier 2009.

[7] Kellerer H, Pferschy U, Pisinger D, *Knapsack Problems*, Springer 2004.

[8] Puterman ML, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley 2005.

[9] Liu C, Bouazizi I, Gabbouj M, "Rate Adaptation for Adaptive HTTP Streaming", *ACM MMSys*, 2011.

[10] Papastavrou JD, Rajagopalan S, Kleywegt AJ, "The Dynamic and Stochastic Knapsack Problem with Deadlines", *Management Science*, 1996.

[11] Winters PR, "Forecasting Sales by Exponentially Weighted Moving Averages", *Management Science* 6(3):324-42, 1960.

[12] "Recommendation BT.500: Methodology for the subjective assessment of the quality of television pictures", *International Telecommunication Union*, 2012.

[13] Wang Y, Schaar M, Chang S, Loui AC, "Classification-Based Multidimensional Adaptation Prediction for Scalable Video Coding Using Subjective Quality Evaluation", in *IEEE Trans. Circuits Sys. Video Tech.*, 15(10):1270-8, 2005.

[14] Zink M, Suh K, Gu Y, Kurose J, "Watch Global Cache Local: YouTube Network Traces at a Campus Network - Measurements and Implications", *IEEE MMCN*, 2008.

[15] Zhou Y, Chakrabarty D, Lukose R, "Budget Constrained Bidding in Keyword Auctions and Online Knapsack Problems", *WWW*, 2007.

[16] Chang SF, Vetra A, "Video Adaptation: Concepts, Technologies, and Open Issues", in *Proc. IEEE*, 93(1):148-58, 2005.

[17] Ha S, Sen S, Joe-Wong C, Im Y, Chiang M, "TUBE: Time-Dependent Pricing for Mobile Data", *ACM SIGCOMM*, 2012.

[18] Rejaie R, Handley M, Estrin D, "Quality adaptation for congestion controlled video playback over the Internet", *ACM SIGCOMM* 1999.

[19] Liu J, Li B, Zhang Y, "An End-to-End Adaptation Protocol for Layered Video Multicast Using Optimal Rate Allocation", in *IEEE Trans. Mult.*, 6(1):87-102, 2004.

[20] MPEG-DASH, <http://dashpg.com/>.

[21] Jan RH, Lin CP, Chern MS, "An optimization model for Web content adaptation", *Computer Networks* 50(7):953-65, 2006.

[22] Onavo, <http://www.onavo.com/>.

[23] Dobrian F, Sekar V, Awan A, Stoica I, Joseph DA, Ganjam A, Zhan J, Zhang H, "Understanding the Impact of Video Quality on User Engagement", *ACM SIGCOMM*, 2011.

[24] Liu X, Dobrian F, Milner H, Jiang J, Sekar V, Stoica I, Zhang H, "A Case fo a Coordinated Internet Video Control Plane", *ACM SIGCOMM*, 2012.

[25] "Global Interet Phenomena Report", *Sandvine*, 2012.

[26] Chen KT, Huang CY, Huang P, Lei CL, "Quantifying Skype User Satisfaction", *ACM SIGCOMM*, 2006.