An Information-Aware QoE-Centric Mobile Video Cache

Shan-Hsiang Shen and Aditya Akella
University of Wisconsin, Madison
{shan-hsi,akella}@cs.wisc.edu

ABSTRACT

Recent years have seen a tremendous growth in the volume of video traffic in mobile settings. In this paper, we present the design of a mobile video-centric proxy cache, named iProxy, that offers improved performance in terms of both hit rates and streaming quality. Our thesis in designing iProxy is that we need to elevate the traditional view of caching from "data" to "information" in order to optimally meet the stringent requirements of video streaming in mobile settings. iProxy relies on recent advances on *information-bound references* (IBRs) to collapse multiple related cache entries into a single one, improving hitrate while lowering storage costs. iProxy incorporates a novel dynamic linear rate adaptation scheme to ensure high stream quality in face of channel diversity and device heterogeneity. Our evaluation of iProxy using realistic traffic traces shows that it can improve hitrate, but we need to use novel information-aware replacement policies for optimal performance. We show that our linear encoder can adapt well to changes in bandwidth, and yield better bit rates, lower buffering and lower start up delays than state-of-the-art schemes.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

Wireless Networks, Caching, Video, Quality of Experience

1. INTRODUCTION

This paper is motivated by two key trends: (1) video-on-demand traffic in mobile settings is seeing unprecedented growth, doubling in volume every few months [2]. Crucially, this is beginning to strain cellular network infrastructures [25]. (2) As user expectations of video quality grow, it is becoming increasingly important to improve the users' quality of experience (QoE). This is central to sustaining subscription/advertisement based revenue generation models which are the drivers behind video growth [2].

In this paper, we focus on how cellular network providers can enhance their infrastructure to meet the goal of ensuring good end user QoE even in the face of growing traffic volumes. One option is to upgrade link capacities everywhere. However, this is expensive and can only provide a temporary solution. Another option is to rely on data offloading techniques, but this may not be universally applicable (e.g., WiFi offload may require open APs).

This paper explores a third approach: in-network caching. Caches are inexpensive, and they can be easily retrofit into an existing cellular backbone. Crucially, while the above approaches improve effective capacity, caches also improve latency which is important for good QoE.

While caching in networks is well-studied, we argue that simply adapting traditional Web cache designs to mobile video is not sufficient. First, in contrast with Web content, video content is often available in multiple different encodings and formats [43]. Not accounting for these differences leads to sub-optimal cache designs that suffer from poor hit rates (up to 20% lower compared with ideal) or need excessive storage (up to 2.4X more). Second, because of the centrality of video QoE, special measures must be adopted to orchestrate the transmission of cache contents to optimize various QoE metrics. In contrast, traditional caches can simply transmit bytes to end users. Our work presents the design, implementation, and evaluation of techniques that address both issues.

Cache design: We leverage the fact that the same video is often available in many different resolutions or formats and from many different content providers. This is particularly true for popular video; indeed, a recent study [43] estimated that nearly 30% of the video search results returned for 25 popular queries at various video content providers were duplicates, i.e., nearly identical to the most popular version of the video, differing only in resolution, encoding and/or the content provider hosting the video.

This insight is embodied in iProxy, our mobile video-centric cache. iProxy derives the information-bound reference (IBR) for each video, which is a collection of summaries of the frequency domain representation of the video data in multiple chunks of the video [17, 33]. We then bind videos that share "identical" IBRs into a *single* cache entry, thereby saving storage. iProxy identifies matches by comparing requested URLs against URLs that are known to have the same IBR, as opposed to looking for URL matches, or matches in content SHA-1 hashes. Because the IBR labels the information content in a video, agnostic of irrelevant details such as the source domain or host providing the content, the format of the content, the presentation, bit rate etc, iProxy enables *caching information as opposed to data*.

We argue that novel information-aware cache replacement schemes must be designed to optimize the hit rate of iProxy. We develop modifications to classical least-recently used (LRU) and least-frequently

MobiCom'13, September 30-October 4, Miami, FL, USA.
Copyright 2013 ACM 978-1-4503-1999-7/13/09 ...$15.00.

used (LFU) schemes that significantly outperform even state-of-the-art information-agnostic schemes.

Optimizing QoE: End-user QoE is reflected in engagement, i.e., fraction of the total video time that a user watches, and abandonment, i.e., whether a user quits a video before it starts [32]. Recent studies [32, 24] showed that a viewer who experiences buffering equal to 1% of stream duration plays 5% less of the video compared with a user with no buffering, and higher average bit rate correlated with higher video play times [24]. Furthermore, users start to abandon a video if it takes more than 2s to start up, and even 1s additional start up delay increases abandonment rate by 6% [32].

We argue that iProxy's information-aware design, coupled with the fact that it is deployed by the cellular provider, offers natural avenues for optimizing the three key quality metrics – buffering, bit rate, and start up delay – thereby improving end-user QoE. With each iProxy entry, we store the raw frequency domain data corresponding to the highest quality video observed thus far across all URLs mapping to the entry, which is needed to compute the IBR for the video any way. We then design a novel linear bit rate adapter that dynamically encodes videos directly from the raw data.

Our scheme first derives client-side constraints by parsing HTTP headers to determine what resolutions and encodings are acceptable; determining these is key to ensuring low start up times (§5). Then our scheme uses information from the mobile device's physical context – which is available to the cellular provider – to determine the baseline bit rate to use, and low-level TCP feedback to estimate network conditions and adapt the bit rate closely to fine-grained channel variations. Our bit rate adapter keeps the average bit rate as high as possible and minimizes bit rate switches, while also controlling re-buffering events. Also, by virtue of being close to clients, iProxy caches offer low RTTs, which translates to higher bit rates, compared with transfers across the network.

We implement a prototype of iProxy using a quad-core Linux desktop. We evaluate it using real traffic traces of handheld video traffic collected over the wireless network of our university over a three-day period and a public dataset of popular video queries at top content providers [6].

Our evaluation shows the following: compared with a naive conventional proxy, iProxy offers a 20% better hit rate for our university trace (60% vs 72%), and improves hit rate by a factor of 1.6X (28% vs 44%) for the public video dataset, compared with a conventional proxy. For our university traces, we find that our new information-aware cache replacement policies offer 15% better relative hit rates than even the best information-agnostic cache replacement strategies. We find that using an information-aware policy is crucial to realizing the benefits of iProxy, and that iProxy uses 2.4X less storage than conventional designs. We present an analysis of where to deploy iProxy in a cellular providers' network. We also conduct a thorough evaluation of iProxy's dynamic encoding and adaptation schemes using an Android smartphone. We show that iProxy improves start-up times by up to 13s by serving a format that is optimally suited for the phone. Some videos can simply not be played on the phone using traditional techniques, but they play almost instantly when iProxy is used. Testing against a variety of scenarios where bandwidth varies dynamically, we also find that, compared with state-of-the-art streaming techniques (specifically, MPEG DASH), iProxy's linear bit rate adaptation improves the average bit rate for a given stream by 16%, and virtually eliminates buffering.

To summarize, the contributions of this paper are as follows:

- We present the design and implementation of iProxy, a cache for mobile video that optimizes end-user QoE while alleviating load on cellular network links.

- A novel design feature of iProxy is that multiple videos containing the same underlying information but differing in format, bit rate or source are stored using a single iProxy cache entry: the index for the entry is an information-bound reference (IBR), and the value is the list of URLs that share the IBR as well as the common underlying frequency domain representation. We show that leveraging IBRs in this fashion can result in 1.2 to 1.6X better hit rates, and up to 2.4X lower storage compared with URL- or content-based approaches. But achieving these results requires careful design of information-aware cache management schemes.

- The second unique feature of iProxy, a novel bit rate adaptation scheme, builds off of the above cache design: it determines the format ideally suited for a device and further uses TCP layer feedback to encode the video directly from the stored frequency domain representation at a dynamically selected bit rate that matches network conditions as closely as possible. This design feature helps improve video start times (by up to 13s), buffering rates and average bit rates (by 16%), thereby improving end-user QoE.

2. MOTIVATION AND BACKGROUND

A recent study [43] analyzed similarity among video results corresponding with 25 popular queries from YouTube, Google Video, and Yahoo! Video [6] and found that on average 27% of the videos are "redundant", i.e., nearly identical to the most popular version of a video in the search results. Simply using URLs will not identify these videos as being redundant because each video has a different URL. In addition, we applied data-centric approaches [36] that identify similarities based on matching SHA-1 hashes of content chunks, but found that these approaches could identify less than a third of the redundant videos; these techniques completely missed out on videos that had minor differences due to resolution, encoding, or quality. Our analysis of requests to video content made from mobile devices traced in our campus wireless network also revealed similar qualitative results (some of these results can be found in §5).

iProxy leverages these observations. It is built on the fact that users may request the same video in different formats or encodings, potentially served by different content providers. In contrast with prior schemes, iProxy's caching scheme can identify redundancy in the face of such access to video because its use of IBRs, which we explain later in this next section, helps it match information contained in a video without being tied to the URL (i.e., the location where it is hosted) and specifics such as encoding or resolution. While we discuss iProxy in the context of caching, we argue in §4.3 that its design is also applicable to *prefetching*.

2.1 Design Requirements

iProxy must meet the following requirements:

(1) Efficient caching: While IBRs help identify matches across multiple versions of a video, we must still understand how to ensure cost-effectiveness of caching, which is reflected in bandwidth savings per unit cost. As the diversity in video formats and sources grows, it is likely that naive approaches that store all redundant versions of a video in every possible format and from every possible source may use a significant amount of storage and, as a result, be quite expensive. We seek a design whose storage cost is significantly lower than this naive alternative, but the bandwidth savings are similar. Meeting this requirement is possible if: (a) iProxy has a way of organizing the cache such that videos carrying redundant information are not stored, and (b) only those entries that correspond to the most heavily accessed information—not data—are stored in the cache.

Good QoE: It is nontrivial to precisely define QoE, because different users may care about different criteria, but in our paper, we focus on start-up latency, video stall time, and video quality (bit rate). Content providers today employ sophisticated mechanisms to ensure QoE, e.g., using third parties to switch downloads across multiple CDNs, and pick alternate bit rates [3]. Because iProxy interposes between content providers/CDNs and end-users, there is a danger that it may undercut content providers' QoE enhancements. To meet QoE requirements, iProxy must ensure that: (a) video is served in a resolution and encoding optimally suited to an end-device so that the start-up delay is as low as possible, minimizing the potential for abandonment [32], and (b) bit rate used stays close to the available bandwidth so that the average bit rate is high and buffering is controlled, maximizing engagement [19, 24, 32]. The use of IBRs in iProxy provides unique opportunities, enabling solutions for both.

When these requirements are met, iProxy can ensure that the effective utilization of providers' network infrastructure can be lowered for a given high aggregate mobile-video traffic load, while ensuring high end-user QoE. In what follows, we describe IBRs which form the basis for iProxy.

2.2 Information-Bound Referencing

Information-bound referencing (IBR) was first introduced in [17]; key details can be found in [33]. IBR derives the information from multimedia data as a "perceptual fingerprint" for the data. This fingerprint isn't tied to a specific protocol, host, file name, or multimedia format such as resolution and bit rate, but is bound only to the information it presents. Algorithms for IBRs have their basis in multimedia fingerprinting, used for copyright violation detection. In this section, we discuss how to calculate the IBR value for multimedia content and how to apply it to iProxy.

IBR for a single frame: We first explain the IBR calculation of a single frame. Prior work [33, 17] presents three options to retrieve fingerprint for an image: leveraging spatial structure, using color distribution, and frequency domain analysis. Of these, frequency domain analysis based on Discrete Cosine transforms (DCT) is shown to be the better choice. The main insight is that the low-frequency components provide a high-level "sketch," and the high-frequency components provide more fine-grained distinctions [16] of the underlying information content. To elaborate, the image is first scaled to a baseline resolution of 128×128 [33]. Then, the YCbCr representation of this scaled image is generated [11]. Discrete cosine transform (DCT) on the Y, Cb, and Cr matrices is then run. The IBR then consists of two parts: First, the low 9×9 frequency components are used, which capture more than 95% of the signal energy. Second, to capture more fine-grained differences, the sum of the high-frequency components of Y is computed; the lower-end 3×3 sub-matrices of Cb and Cr DCT components where most of the signal energy lies are also captured.

IBR for a Video: Prior work [33, 17] suggests videos be chunked and only frames at chunk boundaries be processed to compute IBRs. The way to chunk videos is using scene detection [21] which scans frames for similarity and groups together frames with similar content. The frames at chunk boundaries are called $key\ frames$, where the images change significantly due to scene changes. To identify this scene change, we need to pick a good image feature. Based on extensive experiments, [33] suggests using the variation in the amplitude of the zero-th frequency of the Y component. More specifically, the distance between frames i and $i + 1$ is measured as $DistSeq(i) = \frac{|A_{i+1}-A_i|}{min(A_{i+1},A_i)}$, where A_i is the zero-th frequency of the Y-component of frame i. If the distance is less than $ChunkThresh$, the two frames are considered similar and grouped into the same chunk. [33] suggests that $ChunkThresh$ be set to 0.5. For each chunk, only the first and the last $ImgIBR$s are kept. All $ImgIBR$ pairs of chunks are concatenated to form the IBR value of a video, $VideoIBR$. A 424-byte audio IBR using an existing audio fingerprinting algorithm [22] is also included.

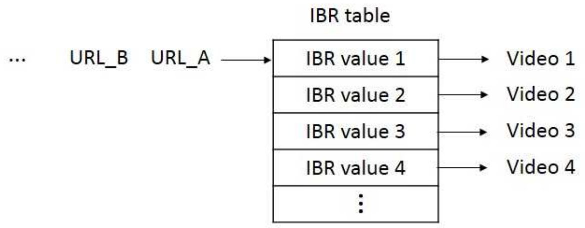

Figure 1: In the IBR table, multiple URLs map to one IBR value, which corresponds to exactly one video.

IBR lookup and lookup performance: As mentioned earlier, the same content may provide slightly different IBRs due to the random noise introduced by different encoding methods. Thus, [17] suggests using fuzzy matching based on locality sensitive hashing [18, 28] to lookup IBRs and identify videos with matching IBRs.

Analysis of IBRs over a large collection of videos in the wild shows that, using conservative match thresholds, IBRs can match related variants with a zero false positive rate. We found zero false positives in our own trace-based analysis (§5). However, false negatives are non-negligible (∼5%) meaning that not all hits will be identified. In addition, [33] shows that IBRs can defend against a variety of attacks on content integrity such as the use of insets (bogus content is embedded), quantization (lower the video quality), and resizing (rescale videos); however, IBRs cannot determine differences arising due to subtitles. Luckily, many modern MPEG4 videos separate video content from subtitles.

We believe that with evolution in multimedia fingerprinting schemes, it will be possible to design even more robust IBRs with lower false negative and zero false positive rates for matching. We use the thresholds recommended in [33] for both deriving and matching IBRs in the rest of this paper.

3. EFFICIENT CACHING IN iProxy

iProxy is located between cellular users and video providers. No modifications are necessary at either end-user devices or video providers; so, for example, end users can still use conventional schemes such as search engines and HTTP requests to peruse and retrieve video content.

To meet the first requirement in §2 iProxy stores an *IBR table* that maps URLs to IBR values. As shown in Figure 1, each IBR value corresponds to exactly *one* video file. Multiple URLs may map to the same IBR value because these URLs represents videos with redundant information, and each IBR value associates with a single video file (no matter how many URLs map to it).

After receiving a request for a URL, iProxy checks the IBR table first (Figure 2); this table maps an URL to a particular IBR value. Three cases arise:

The URL hits: The URL requested by an end user can be found in the IBR table, which means that this exact URL was requested

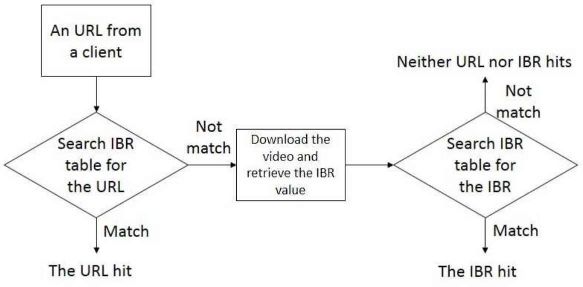
An URL from a client
Search IBR table for the URL
Not match
Download the video and retrieve the IBR value
Search IBR table for the IBR
Not match
Neither URL nor IBR hits
Match
The URL hit
Match
The IBR hit

Figure 2: The flow of cache matching.

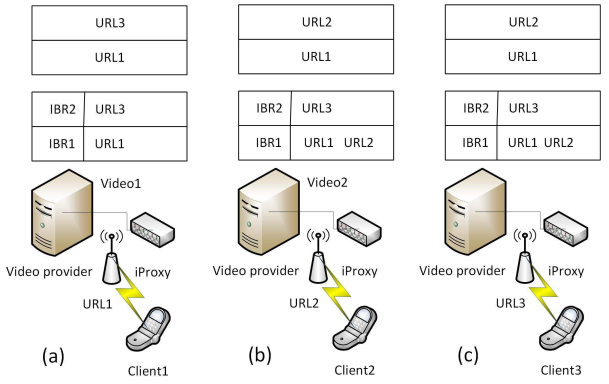

Figure 3: iProxy vs traditional cache.

previously. In this case, iProxy can transmit the cached video to the end user.

The IBR (eventually) hits: When the requested URL cannot hit any entry in the IBR table, iProxy forwards the request to the corresponding content provider. The provider transmits the requested video, and iProxy transforms it into frequency domain data. The transmission and transformation can be processed in parallel to reduce overhead. iProxy then derives the IBR from this frequency domain data as described in §2.2, and the computed IBR is then looked up. A match means that the same content was requested earlier, but from a different source and/or in a different format. In this case, iProxy compares the quality of the downloaded video against the in-cache video corresponding with the matching IBR. iProxy keeps the higher-quality version and drops the other (assuming the cache is full, which is iProxy's steady-state mode of operation). Finally, the IBR table is updated by inserting the new URL into the existing entry corresponding with the matching IBR. Using this approach ensures that with a limited cache size, iProxy can cover more and more URLs over time than a conventional proxy, which means its cache can satisfy more requests.

Neither URL nor IBR hits: After video transmission from the provider and frequency domain transformation, the computed IBR may not hit any entry in the IBR table. In this case, a new entry created for the IBR is inserted, the requested URL added to the entry, and the frequency domain data for the video is cached. The new entry and the corresponding video need to replace one or more IBR entries and corresponding videos in the cache (discussed in §3.1).

Example: Figure 3 compares iProxy to a conventional proxy. In this scenario, there are two smartphones called $Client1$, $Client2$, and $Client3$ which connect to a cellular tower. iProxy is behind the cellular tower. Assume the cache size is bound to two entries. The upper table is the cache (index) of a conventional proxy, and the lower table is the IBR table of iProxy. Both employ FIFO replacement.

In Figure 3(a), $Client1$ requests a video with $URL1$. $URL1$ is not in the IBR table, so iProxy forwards the request to the provider. After receiving a video named $Video1$, iProxy calculates its IBR value as $IBR1$, which does not exist in the IBR table. Thus, the mapping between $URL1$ and $IBR1$ is inserted, and $Video1$ (its frequency domain representation) is cached. Then, the video is forwarded to $Client1$ via dynamic video encoding. A conventional proxy works similarly, but does not employ dynamic encoding or adaptive video rate adjustment, of course.

Next, in Figure 3(b), the $URL2$ request from $Client2$ is not in the IBR table either. Thus, iProxy fetches $Video2$ from the video provider and calculates its IBR value, $IBR1$, which is close enough to the IBR value of $Video1$. Thus, we can say that $Video1$ and $Video2$ include the same underlying information. iProxy prefers to keep videos with higher quality and drops the others. In this example iProxy decides to keep the higher-quality $Video1$ and drops $Video2$. Finally, $URL2$ is inserted into the IBR table to show that $URL2$ also maps to $IBR1$. In this case, iProxy does not replace any entry and all of $URL1$, $URL2$, and $URL3$ stay in the table. However, a conventional proxy replaces potentially valuable $URL3$ content by $URL2$; it also wastes storage in caching the content for both $URL1$ and $URL2$, although they both refer to the same underlying information.

Later requests for any of $URL1$, $URL2$, or $URL3$ will hit in iProxy, but $URL3$ doesn't hit in a conventional proxy. For example, suppose $Client3$ requests videos with $URL2$ and $URL3$. Both $URL2$ and $URL3$ are in the IBR table and map to $IBR1$ and $IBR3$, respectively, so $Video1$ and $Video3$ will be forwarded to $Client3$. However, a conventional proxy can only hit $URL1$ and needs to request $Video3$ for $URL3$ from the provider.

3.1 Info-aware Cache Replacement

While the aforementioned design ensures that redundant videos are not stored, its FIFO cache management policy does not provide control over the usefulness of the videos stored. In this section, we consider how to design cache management schemes that work with the rest of iProxy's design, to store the most valuable data and keep hit rates high.

To optimize hit rate, our schemes prefer entries whose corresponding information shows both sufficient locality of access as well as wide-spread coverage across many URLs. In particular, we modify the least frequently used (LFU) and least recently used (LRU) schemes to result in two new schemes, as follows:

Our modification to LFU is called "LFU-based IBR-score" policy wherein we give each IBR table entry a score defined as $score_i = \frac{\sum_{u \in URLs}(\frac{C^u_{hit}}{C^u_{stay}})}{VideoSize}$. $score_i$ is the score of $entry_i$ and $\frac{C^u_{hit}}{C^u_{stay}}$ is the sub-score for individual URL u which references the IBR value that $entry_i$ represents. C^u_{hit} is the "hit count", i.e., the number of hits for URL u and C^u_{stay} is the "stay count", i.e., how long URL u has been in cache in terms of the number of total requests arriving at the cache. Thus, $\sum_{u \in URLs}(\frac{C^u_{hit}}{C^u_{stay}})$ captures the contributions of different URLs u to the overall "interest" in the information represented at $entry_i$. We divide the frequency by video size, so shorter videos tend to get higher score.

The policy then is to evict the item with the lowest score. This policy is better than naive LFU: unlike LFU, it does not have the "cache pollution" problem[1] because it considers how long an entry stays in cache (C^u_{stay}).

This scheme is better than an alternative that ignores per-URL hit and stay counts and instead computes the aggregate hit/stay counts for the IBR table entry as a whole. This alternative is a direct generalization of cost-aware cache replacement schemes such as Greedy Dual Size-Frequency (GDS) [23][2] that have been shown to be more effective than LRU and LFU. Our scheme is better because it is *information-aware*. In particular, it simultaneously prioritizes frequently referred entries as well as entries that cover more URLs. While the former (number of accesses per IBR table entry) captures the relative interest in a certain information, the latter (number of URLs for an entry) captures both the many different representations for information (i.e., different encodings/formats, which are often given different URLs at a given content provider) and ways of accessing the information (i.e., different content providers). In doing so, our scheme also better accounts for temporal locality that arises when a small number of representations or content providers are repeatedly employed when a certain piece of information is accessed. We empirically establish the superiority of our scheme in §5, especially in comparison with LFU and GDS.

We also design a modification to LRU called "LRU-based IBR-score" and it computes the following: $score_i = \frac{\sum_{URLs}(\frac{1}{C_{last}})}{VideoSize}$. C_{last} means how long the last hit for a URL has been in the cache in terms of the number of total requests to the cache. $\frac{1}{C_{last}}$ is the sub-score for an individual URL. Higher $\frac{1}{C_{last}}$ means this URL is hit more recently. As the earlier scheme, this simultaneously prioritizes most recently referred entries and those that cover more URLs.

3.2 Cache Deployment

In a canonical cellular network architecture, there are a few potential locations to deploy iProxy, each with different trade-offs: at the radio network controller (RNC), at the serving GPRS support node (SGSN), and at gateway GPRS support node (GGSN). The RNC is closer to the user equipment (UE), so the latency between RNC and UE is lower, which can be important for QoE, discussed next. However, being low in the hierarchy, each RNC may only serve a few users, which may, in some cases (where population density is low), result in low hit rate. At the other extreme, GGSN aggregates many UEs so it can provide a higher hit rate, but this comes with much higher latency, which affects QoE.

Our design permits deployment at all three locations in the hierarchy. The "optimal" location depends on the cellular network subscriber density and the cache location relative to content servers. We evaluate these trade-offs in §5.

4. OPTIMIZING QOE

As mentioned earlier, iProxy must ensure good end-user QoE as it transmits videos out of the cache. There are two aspects to this: ensuring high bit rates and low buffering rates, which maximize user engagement, and ensuring low start-up times, which minimizes abandonment. We discuss these in the following two sections. In both cases, we argue that iProxy's information-aware design discussed earlier, coupled with the fact that it is deployed inside the cellular network infrastructure, provides unique opportunities.

Many cellular providers already employ transcoding proxies [20] that transform Web content to a suitable format for mobile device users to save bandwidth. For example, such proxies are employed to reduce image resolution for smartphones. However, we are not aware of any proxy that dynamically adapts video according to channel quality. That said, iProxy can work in conjunction with such proxies.

4.1 Bit Rate and Buffering: Handling Channel Diversity

Several prior works have measured the throughput, jitter, and loss rate of 3G networks and have found that the channel conditions and, hence, throughput performance, vary widely with location [26] and also over time at a given location [26, 38]. The dynamic range can span a few tens of Kbps to a few Mbps, and the variations can happen on sub-minute timescales [38].

Ensuring high QoE implies adapting the video stream to such rapid, arbitrary changes in bandwidth as channel conditions change and/or the mobile device moves.

In what follows, we describe a novel approach for dynamic video encoding, which drives all video transmissions out of the iProxy cache. It does not require the active participation of CDN servers or video content providers nor does it require modifications to clients.

Before describing our framework for addressing these issues, we start by providing some background on MPEG 4-based video encoding, which we use as the basis for our dynamic encoder.

MPEG4 Video Encoding - Background: Figure 4 presents a typical MPEG 4 video encoding process. The whole process can be divided into discrete cosine transform (DCT), scaling/quantization, motion estimation, and entropy coding.

DCT computes frequency domain data for video content [15]. More important data is located in low-frequency components and data, including more detail located in high-frequency components. DCT is widely used in lossy compression for multimedia content. Quantization determines the bit rate of the video stream. The quantization process follows DCT and compresses a range of continuous values into one discrete value. Usually we give more bits to represent low-frequency components, which are more important, and fewer bits to represent high-frequency components, which are less important. By choosing the quantization strategy, we can determine the bit rate.

Motion estimation can further reduce video size. Before compressing a video frame, we cut a video frame into multiple blocks and try to match these blocks with other blocks in the same or previous or following frames. If some blocks can be matched, we keep only their reference pointers (also called motion vectors) instead of the blocks themselves. Frames can be categorized into I-frames, P-frames, and B-frames. I-frames do not refer to any frames, P-frames refer to earlier frames (usually I-frames), and B-frames refer to earlier or following frames.

Finally, we code all data including the output from quantization and motion estimation using entropy coding. Entropy coding gathers statistics about how often each symbol appears. High-frequency symbols are given shorter codes for representation. Thus, on average the code length is reduced, which decreases the video size.

4.1.1 Dynamic Video Encoding: A Linear Bit Rate Adapter

We argue that providing high QoE in the face of cellular channel diversity implies that iProxy should provide a flexible video bit rate that is as close to the available bandwidth as possible. State-of-the-

[1] Since LFU only considers hit counts, entries that are very popular briefly tend to stick around in the cache

[2] GDS calculates a score which is the ratio of cost and size ($\frac{cost}{size}$) for each entry. The $cost$ is measured in reference counts (i.e., frequency). $Size$ is the data size. When the cache is full, GDS evicts the entry with lowest score.

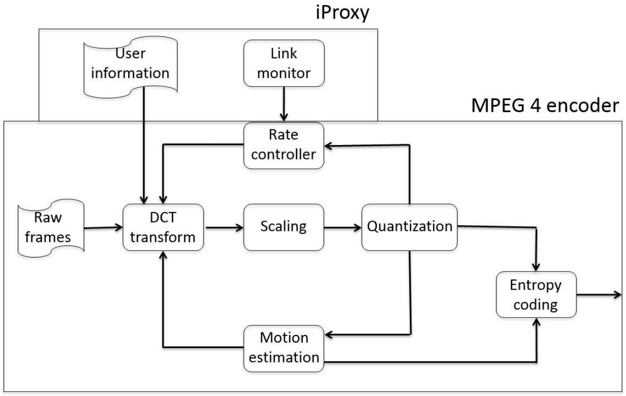

Figure 4: MPEG4 encoding overview, and how iProxy adapts the encoder.

art solutions, e.g., layered video encoding or MPEG DASH, are not suitable, as they cannot offer linear bit rate adaptation, e.g., MPEG DASH it employs k different pre-selected versions of the video, selecting the appropriate version over time based on client-reported performance metrics. Unfortunately, these versions may differ significantly in the bit rate and the quality[3]. While using large values of k, where the corresponding videos are closer to each other in quality, enables smoother adaptation, it also has downsides: it needs nearly $k\times$ more storage in iProxy, and it is hard to predetermine a set of k encodings that will work across all possible mobile clients. Also, the bit rate adaptation scheme in DASH works on the time-scale of several seconds [30], due to which there are multiple time-instances when the bit rate and available bandwidth differ, especially under dynamically changing conditions. As we show in §5, these attributes can cause the stream to use a lower-than-ideal bit rate or overshoot available bandwidth significantly, resulting in stalls/buffering, both of which result in poor QoE.

iProxy uses a new dynamic video encoding scheme to provide *linear bit rate adaptation*. It uses in-band schemes that provide fine-grained information on channel conditions, such that the resulting stream can gracefully fit the available bandwidth, meaning there are few buffering events, if any, and the video quality stays as high as possible. Different from other mobile video schemes, iProxy doesn't need any modification on the client and video provider side. In addition, it doesn't rely on physical layer support. Consequently, it is easier to deploy in existing infrastructures.

At a high level, the linear bit rate adapter works as follows: the original MPEG 4 encoder uses a rate controller to achieve a constant bit rate. The rate controller takes a particular bit rate and feedback from the quantization module as inputs. Then, it calculates the appropriate parameters for the quantization module to output video stream with a fixed bit rate. iProxy adds a link monitor module to monitor available bandwidth and send this information to the rate controller in the MPEG 4 encoder, as shown in the top box in Figure 4. We modify the rate controller to accept the bit rate suggested by the iProxy link monitor on-line. iProxy's link monitor uses two schemes to determine the network conditions in order to pick a bit rate, which we will describe shortly.

To enable fast dynamic video encoding based on the suggested bit rate, we cache frequency domain data and motion estimation vectors for each video instead of caching encoded videos. This allows us to skip calculating the DCT when performing dynamic video encoding. To facilitate this, when receiving a video from a video provider, iProxy transforms the video into the frequency domain data and retrieves its motion estimation vectors. As shown in §2.2, frequency domain data is also needed during IBR value calculation. Thus, caching frequency domain data and calculating the IBR value can be done in parallel.

Baseline bit rate using in-context information: The first scheme operates on coarse time-scales and establishes a baseline bit rate to use. The baseline is reset/chosen each time there is a significant change in the mobile device context, in particular, when the device moves to a different cell tower (known to the cellular provider), meaning that the cache-to-device network path is now different and we need to adapt the stream to it. The baseline for each cell tower can be computed by the cellular provider based on historical measurements of the average throughput mobile devices associated with the tower observe at a given time in the day.

TCP information feedback: Once a reasonable baseline is chosen, we must then adapt the bit rate as the achievable throughput varies on fine timescales. For this, we rely on per-packet TCP feedback information. In particular, iProxy collects current TCP-level information for a stream, such as the congestion window (CWND) and RTT (by reading off the TCP/IP stack). iProxy then estimates the current available bandwidth as: $Available\_bw = \frac{CWND \times segment\_size}{RTT}$. Finally, iProxy uses an exponentially-weighted moving average (EWMA) of $Available\_bw$ to compute the bit rate to suggest to the MPEG4 rate controller in Figure 4. Using EWMA helps avoid rapid bit-rate shifts. We set the weight to 0.9 to ensure smooth bit rate changes. Also, to account for changes in the device's cell tower, we ignore the available bandwidth reported for the first few RTTs after a device has moved and rely on the baseline bit rate instead. Finally, when the expected bit rate does not exactly match real network bandwidth, we rely on buffering at the client side to smooth out the videos.

In §5, we use extensive experiments to show that our bit rate adaptation schemes enables iProxy to keep the bit rate as high as possible, while also eliminating stalls.

4.2 Low Start-up Time: Accommodating Client Diversity

A recent study of handheld traffic in a campus network [27] showed that users employ devices from many vendors and with different form factors. The advent of tablets only exacerbates this diversity. The challenge is that each model typically has a different screen size and supported resolution. Thus, videos with different resolutions may be requested at iProxy. The study found a wide diversity in device OS, which means that the video decoders the devices have installed could be quite different. Finally, the study found a wide diversity in video formats requested.

A key challenge in iProxy is to stream compatible videos to clients in the face of such diversity. This is crucial to optimize start-up time: as we show in §5, transmitting a video that is encoded at a higher resolution than a device's screen resolution can inflate start up time by 2-14s; and, picking the wrong encoding can result in the video being unplayable (i.e., the start up time is infinity).

In order to stream compatible videos, client-specific information must be obtained by iProxy's dynamic video encoding module. We discuss how this can be done next.

OS: A majority of mobile devices employ Android, iOS, or Windows mobile. This can generally be inferred from the HTTP header, which often includes the operating system version and device type.The videos each operating system can handle are not the same; e.g., iOS

[3] Four versions that offer 38 dB, 42 dB, 47 dB, and 50 dB in peak signal-to-noise ratio (PSNR), require 800 kbps, 1200 kbps, 1600 kbps, and 2000 kbps of bandwidth, respectively

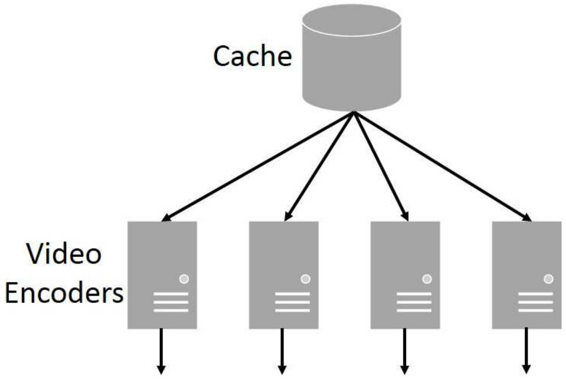

Figure 5: Multiple dynamic video encoding nodes can be added to improve scalability.

cannot support flash-related formats. iProxy tracks this information.

Screen resolution: Common screen resolutions of smartphones vary from 240x320 to 720x1280. When a smartphone is sent a video at a resolution larger than the phone can display, two problems can arise: first, the smartphone has to spend a significant amount of time buffering the high-resolution stream compared with using the optimal resolution, which inflates start-up delays (§5); second, the video cannot be displayed optimally on the screen, which further has an impact on the user experience. earlier, we can retrieve device models from the HTTP headers, and the screen resolution of any particular model is fixed. This is the upper bound of the resolution that iProxy should provide to the end user. In other words, we may reduce the resolution of a video requested by a client if her screen cannot support high-resolution videos.

Decoder: iProxy should ascertain what end users can decode. Some operating systems support built-in video decoders; e.g., Android 2.3.3 supports 3GPP (.3gp), MPEG-4 (.mp4), MPEG-TS (.ts), and webM (.webm) [1]. Thus, the OS can give us a hint as to what video format can be decoded. However, end users may install other decoders. To infer this, cellular providers can track requests sent to to app stores to estimate what other decoders clients have installed and, thus, the encodings/formats of videos the clients can decode and play.

4.3 Applicability to Prefetching

While iProxy operates as a cache, it is also applicable to prefetching which is becoming popular in the mobile video context. In particular, the cellular provider can prefetch high quality versions of videos that users are likely to access and store the corresponding frequency domain representations in iProxy, just as above. For each entry created in this manner, the provider can also obtain a list of URLs where alternate versions of the video are stored; this could be done by crawling the Web and identifying duplicates of the stored entries using the IBR matching algorithms described in §2.2. Subsequently, iProxy operates just as described above both in terms of how storage is managed and how content is streamed.

4.4 Scalability

A natural question that arises is whether this design is scalable: in particular, can the dynamic encoding scale with number of end-users. In §5.4.2, we show that dynamic-encoded videos are on average 42 times shorter than the original video's length, so one single machine can support multiple clients at the same time. To support even more clients simultaneously, we can leverage parallelism, as shown in Figure 5. Here, a single cache tries to match requested videos and download unmatched videos. In addition, there are multiple nodes that work on dynamic video encoding. Encoding workload can be spread among these nodes. All requests are sent to the cache first and, after obtaining the requested videos, each video is redirected to one of the dynamic video encoding nodes. The dynamic encoding nodes finally transmit videos to clients with a specific video format. Cellular network providers can provision dynamic video encoding nodes based on expected total load.

5. EVALUATION

Our evaluation addresses the following issues: (a) **Caching**: How do different replacement policies compare and how important is info-awareness? How much cache storage is needed? (b) **Handling diversity:** To what extent does selecting the right resolution and encoding help improve start up times? (c) **Dynamic encoding**: How well does dynamic encoding adapt to changing conditions? Does it lead to improvements in quality metrics such as buffering rate and bit rate that impact end-user QoE?

We first describe our prototype implementation.

5.1 Implementation

On the client's side, we use *unmodified* Android smartphones, running VPlayer [10]. VPlayer can send a request with a URL to iProxy through 3G.

We implemented iProxy in a 4-core desktop with Intel(R) Core(TM)2 Quad 2.66GHz CPU Q6700 and 8GB RAM. Our prototype proxy, written in C, can fetch videos from video providers, calculate IBRs, compare IBRs, cache raw video data, match videos, and dynamically encode videos. We modified the pHash library [7] to calculate IBRs.

For each request, our proxy iProxy checks its IBR table; we use a locality-sensitive hashing (LSH) [29] based index to aid fast lookups at scale. If the URL is not in the table, iProxy downloads the video from a original website and redirects it to the client. After the video is downloaded, we derive its IBR, and compare this IBR to others in cache (using LSH to retrieve candidates, and then obtaining nearest matches). If we find a matched IBR, we delete the copy with lower quality and update the IBR table by adding the new URL.

The video is then sent to a modified FFmpeg [4] module that supports our dynamic video encoding. It interprets requests from clients to determine client-side constraints. FFmpeg uses FFserver [9] to stream the video to VPlayer in the client's device. During streaming, the FFserver module retrieves dynamic CWND and RTT by reading the relevant network stack variables and calculates suitable video bit rates to use. It then sends this information to FFmpeg which adapts bit rate.

5.2 Traces

Our analysis of iProxy relies on two sets of real traffic traces. Our first trace is the Web video data set [6] we used the motivating statistics in §2. This data set had a total of 10,000 videos corresponding to the search results for the top 25 queries at three popular video content providers. This amounts to about 400 URLs per query: note that the videos corresponding to some of the URLs corresponding to a query may point to the same underlying information, whereas others may correspond to "related" videos that have entirely different information altogether. The total size of the videos is about 300GB.

In addition, we leverage packet traces collected over the University of Wisconsin's Wireless Network. The traces span three days, from April 26, 2010 (Monday) to April 28, 2010 (Wednes-

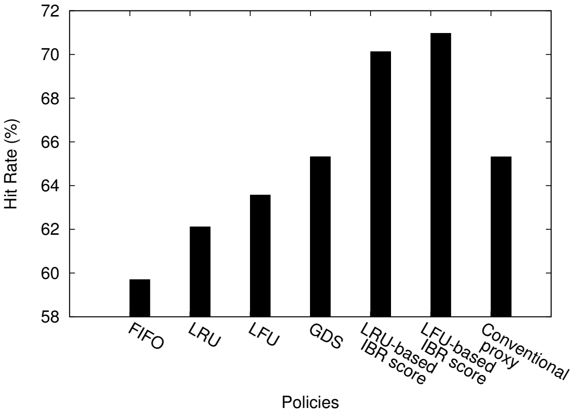

Figure 6: Hit rates for various cache policies in iProxy.

day). We prune the traces to only capture the subset of devices that are known to be smartphones or tablets, using the techniques identified in [27]; the techniques have known false negatives (but minimal false positives), meaning that we don't capture all handheld devices. Furthermore, we focus mainly on the trace subset containing the HTTP protocol. For each packet, we collect the first 128B, which includes the HTTP header.

We identify video traffic based on the "content type" field in the HTTP header, similar to [27]. We derive URLs from the HTTP requests for videos, and download the corresponding video files in their entirety.

5.3 Caching: Hit rates, Storage, Deployment

Replacement Policies: To evaluate the performance of our IBR specific cache replacement policies, we can compare it with other traditional cache replacement policies, including FIFO, Least Recently Used (LRU), Least Frequently Used (LFU), and Greedy Dual Size-Frequency (GDS) [23]. Unless otherwise specified, we assume that the underlying cache is IBR-based. We use hit rate (HR) as the main metric to compare the policies. All caches have the same 1GB size. We use the traces described above for the analysis.

Results from our analysis of the university data set are shown in Figure 6. As expected, FIFO offers the worst performance on HR, because it doesn't consider the importance of cache entries. The HRs of LFU and LRU are around 62%. In contrast, info-aware approaches (modified LRU or LFU) result in higher HR (70-71%), which is better than the state-of-the-art GDS scheme. The difference between GDS and our schemes is that the former only accounts for the overall interest in information, but not the number of different avenues for accessing it; both aspects must be taken into account when designing an information-aware scheme. GDS could evict large collections of URLs in bulk, which may impact future hit rates.

Interestingly, the conventional proxy offers same or better hitrate than an iProxy that uses an info-agnostic replacement policy such as FIFO, LRU of LFU. This is because it is a sub-optimal policy that is not aware of the importance of information as captured by the hit rate across multiple related URLs. This shows that we need to use info-aware replacement policies to realize the benefits of iProxy.

The rest of the paper assumes the LFU-based IBR-score policy is employed in iProxy.

Storage Requirement: Recall that iProxy stores the frequency domain (DCT) data for the highest quality video in each IBR entry. What storage cost does this impose? As exemplified in Table 1, DCT data is larger than individual video files (we show only four different bit rates for a single example video), but: (1) the difference is not significant compared to any single video file (2.1X in the worst case) and (2) the DCT data size is 2.4X smaller than the sum of the sizes of the individual video files, meaning that iProxy outperforms the naive strategy of storing multiple versions.

Video bit rate	Size
885 kbps	3744 KB
1063 kbps	4588 KB
1277 kbps	5403 KB
1385 kbps	5862 KB
DCT data	8154 KB

Table 1: For a single example video, we show the size of the raw data stored with an iProxy cache entry, vs. that of different formats of the video file.

Population	RNC	SGSN	GGSN
1400	1373.55 kbps	1303.9 kbps	1271.93 kbps
800	1282.18 kbps	1259.99 kbps	1176.55 kbps
500	1244.51 kbps	1194.67 kbps	1182.06 kbps
300	1190.22 kbps	1165.02 kbps	1123.62 kbps

Table 2: Throughput vs. location and population.

Deployment Evaluation: As mentioned in §3.2, the performance of iProxy depends on where we deploy it and what the served population is. If we put iProxy closer to clients, the latency is lower and the performance is better. However, a less served population means less overlap in accesses which affect hit rate. We study this question using an emulation based on our traces.

To understand this, we first measure the throughput between a machine at our university and the nearest YouTube server, which determines the cost of a cache miss and the relative improvement from a hit. We calculate mean and variance and model the cost as a normal distribution. In addition, we use the data in [39] which shows the delay to RNCs, to SGSNs, and to GGSNs from UEs in a tier-1 cellular network.

We identified a total of 1400 unique users in our traces. We divide them into smaller groups and associate each group with a node in the same level of the cellular hierarchy. We compute and report the average throughput across all groups. The results are shown in Table 2. In this table, the entry corresponding to row "500" and column "SGSN," means that we divide the 1400 users randomly into three groups of 500, 500, and 400, and "assign" them to three SGSNs. We emulate requests of UEs in each group, identify cache hits/misses and compute the effective throughput observed by the clients. Each miss goes to the YouTube server, whereas a hit is served from cache.

Our goal is not to show that one location is clearly better than the other, but rather to show the trade-off between population density, iProxy location and performance, helping the cellular provider make the appropriate choices. In general, our results show that when the population density is low (e.g., 300 users per RNC, 800 or more at SGSN, and 1400 or more at GGSN), aggregating a small number of iProxy's at the SGSN and GGSN is reasonable. When the population density is high (e.g., 1400 users at an RNC), deploying at the RNCs is a good choice.

5.4 QoE

The setup we use to test iProxy's ability to support good QoE is shown in Figure 7. We stream videos from an iProxy located on our campus to mobile phones also located close by. However, the

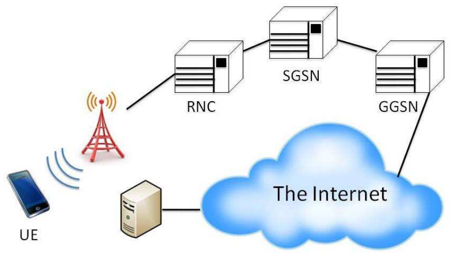

Figure 7: Experimental scenario for video performance.

	Daily-motion	YouTube (HD)	Yahoo Video (.asf)
Both hit	0s	13s	∞
iProxy hit	2s	14s	∞

Table 3: Improvement in video start up latency using iProxy and a conventional proxy. ∞ means the client cannot play the original video format, but can play re-encoded video from iProxy.

transfers themselves traverse the Internet and the cellular backbone before reaching the end device. In practice the proxy would be deployed a lot closer to clients; thus, our experiments below show a lower bound on the effectiveness of iProxy in ensuring QoE.

5.4.1 Start up times

We setup an experiment to show how iProxy improves start up latency. Here, we assume a set-up where users click on embedded links to videos, e.g., links in emails, blogs, etc.

In Table 3, we stream three different videos to an Android smart phone (Samsung GALAXY SII) with a 480×800 screen. The first video is in VGA (640×480 resolution, .flv) from Dailymotion, the second one is in XGA (1024×768 resolution, .flv) from YouTube, and the third one is in 360×288 (.asf) resolution from Yahoo! Video. The first column shows the improvement in video start up latency in seconds in the case where both iProxy and a conventional proxy observe a cache hit for the video request. The second column shows the improvement when only iProxy observes a cache hit.

The video from Dailymotion is in a suitable format and resolution for the client's device, so when both proxies see a cache hit, iProxy does not offer any improvement. However, when a conventional proxy does not see a cache hit, it takes two extra seconds to retrieve the video from Dailymotion, delaying start-up correspondingly.

The resolution of the second video from YouTube is much higher than the smartphone can play, causing the phone to spend time filling its cache and pre-processing the video. Furthermore, because of the resolution mismatch, the XGA video appears pixelated on the smartphone. iProxy lowers start up latency by 13s; it also converts the video into a more suitable resolution (VGA).

The smartphone simply cannot open and play the third (.asf) video file. iProxy converts the video into a suitable format (.mpg), so the smartphone can still play it.

5.4.2 Bit rate and Buffering

Speed: To support user video playback without lag, the time needed to dynamically encode a video from raw data should be small, otherwise users may face stalls waiting for encoded video data. To study this, we use modified ffmpeg to encode video into different bit rates to observe encoding speed compared to video length in seconds. We chose a 590s video and encode it with bit rates from 200 kbps to 1000 kbps. As shown in Table 4, encoding times are similar no matter the bit rate, and are 42 times shorter than the video length. This provides evidence that dynamic video encoding can work on-line, matching real-time video playback requirements. This is further evidenced by our experiments below.

Video length	586.98 sec
200 kbps	13.34 sec
400 kbps	13.94 sec
600 kbps	14.03 sec
800 kbps	14.36 sec
1000 kbps	14.54 sec

Table 4: Video length and encoding time

To measure the efficacy of our linear bit rate adapter in improving QoE, we experiment with scenarios where we estimate how well iProxy functions with rapid changes in network conditions. In each scenario we first send a video around 50 seconds long with a starting bit rate of 800 kbps to a smartphone over 3G. Every few seconds, a bandwidth shaper kicks in at the desktop where iProxy is running to change the available bandwidth according to a predefined pattern. Note that the available bandwidth is also affected by channel diversity; this is not in our control. We measure the average bit rate received by the smartphone and the extent of buffering, both of which impact engagement.

We compare our linear adapter against a version of iProxy that uses the state-of-the-art MPEG DASH scheme. Note that this scheme does not use cellular phone context. Also, it employs k different versions of the video (§4.1). While using large values of k enables greater adaptation, it also uses more storage. To strike a balance, we select $k = 4$, which uses nearly 2X more storage than iProxy with the linear adapter.

We tried out several different scenarios, each with a different way in which bandwidth gets shaped. We shows results for two randomly picked scenarios below.

Note that FFmpeg reports video bit rate of encoded video every two seconds.

Figure 8(a)shows the change in video bit rate in Scenario 1. In the beginning, the available bandwidth out of the desktop is 1000 kbps and video bit rate used by our linear adapter (Figure a) varies between 700 kbps to 1400 kbps. The variation is caused by uncontrolled background traffic. Twenty seconds later, when the bandwidth is shaped to 500 kbps, our linear adapter can detect the change and reduce the bit rate to around 400Kbps to avoid frame loss; our linear adapter keeps the bit rate at 400 kbps. Note that despite the use of EWMA, our linear adapter adjusts bit rate almost immediately after TCP detects packet loss. However, because of the use of a buffer in clients, these bursts in bit rate do not cause any perceptible impact to the user. When available bandwidth drops further to 300 kbps, it is detected by our linear adapter based on the smaller average CWND, causing the linear adapter to continue to decrease the bit rate. Because TCP CWND can reach available bandwidth more quickly when bandwidth is low, the bit rate used by the linear adapter can more closely match the available bandwidth. On the whole, the average bit rate used by our linear adapter is 490Kbps.

Figure 8(b) shows the bit rate used by MPEG DASH. While DASH can also adapt, we see that its bit rate is often significantly lower than the available bandwidth, due to the discrete choices available (e.g., between 40s and 60s). The average bit rate of DASH is about 430Kbps; our linear adapter's bit rate was 16% higher on average. Between 40s and 60s, our linear adapter's bit rate was twice as high as DASH (200Kbps vs 100Kbps).

Figures 9(a) and (b) show the results for a second scenario where the shaper causes oscillations in the bandwidth. For the linear

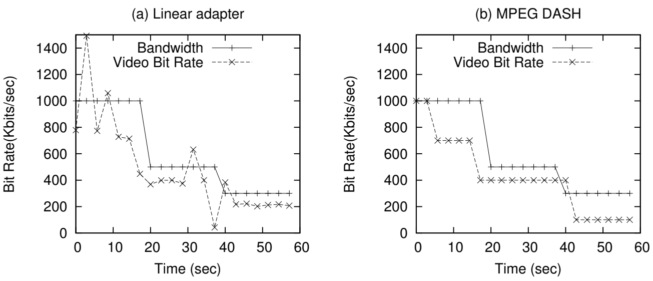

Figure 8: Scenario 1: the available bandwidth set by the shaper falls over time. The left figure is for iProxy using the linear adapter, the right is using MPEG DASH.

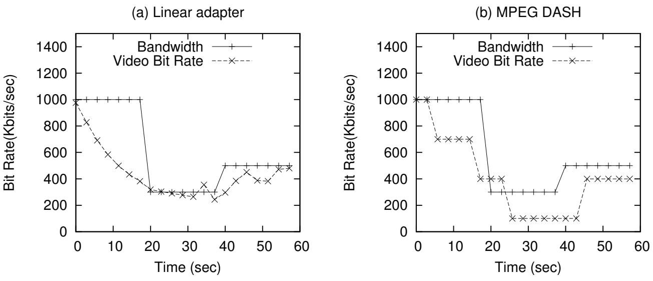

Figure 9: Scenario 2: the available bandwidth set by the shaper oscillates. The left figure is for iProxy using the linear adapter, the right is using MPEG DASH.

adapter, we observe that the bit rate gradually falls between 0 and 20s, perhaps because of poor channel conditions which cause the true available bandwidth to be much lower than the 1000Kbps limit set by our traffic shaper. Our scheme starts by using an statically picked initial rate of 1000Kbps; however, if the initial rate were set based on an accurate available bandwidth measure such as that inferable from the latency spread observed for TCP ACKs, we would have picked a better initial bit rate. In general, relying on direct available bandwidth estimates in this fashion could help our bit rate adapter pick better rates even during the course of streaming, compared to our current scheme of using CWND/RTT; we plan to extend our adapter to work in this fashion in the future.

At around 20s, the bit rate settles to approximately 400Kbps. After 40s, the traffic shaper sets the bandwidth to 500Kbps, and our linear adapter immediately adapts its bit rate to around 450Kbps. In contrast, MPEG DASH generally uses a lower bit rate than our linear adapter (410Kbps, on average), especially in the 20-60s time period where it is nearly 40% lower.

In both cases, our scheme suffered from *no* buffering events. In contrast, MPEG DASH saw up to 1s of buffering. While this may seem insignificant, prior studies [32] show that this lowers engagement to levels that start to matter to content providers.

Sensitivity Because iProxy estimates available bandwidth using simple passive measurements, there is a possibility of error, especially when the available bandwidth drops suddenly. To understand how this impacts QoE, we use the same setup as above, but shape bandwidth such that it drops suddenly from 2000 kbps to 400 kbps roughly 11s into the video (total length of 40s). We observe that the video stream suffers from around 1s of buffering in all. While this is not conclusive, it indicates that iProxy's adaptation scheme is reasonably robust against sudden variations.

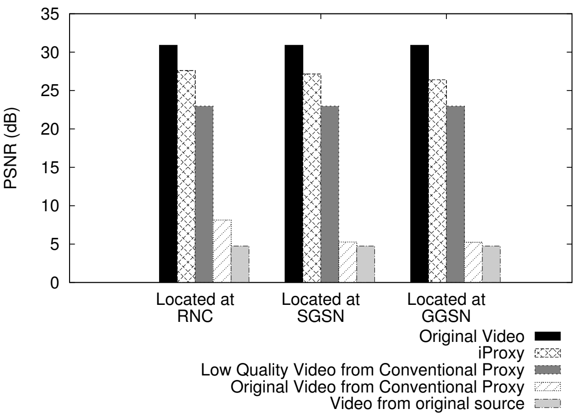

Figure 10: Video quality evaluation

For completeness, we also study a more traditional quality metric, namely Peak signal-to-noise ratio (PSNR). PSNR is calculated by comparing the difference in quality between raw videos and compressed videos. Higher PSNR means higher quality. There are two aspects that affect PSNR: (a) *frame loss:* if some frames are dropped, much of the information in the video will be lost, and (b) *the bit rate:* lower bit rate means we lose more information. We use an emulation-based study that also models cache deployment alternatives.

For the test video we used above, the video is compressed with 30.9dB in PSNR and 2.53Mbps in bit rate; thus, these are the maximal PSNR and bit rate we can achieve with iProxy's dynamic encoding module. We emulate three scenarios representing situations where proxies are located at the RNC, SGSN, or GGSN. We assume network bandwidth is 1.5Mbps. The average delay from the RNC to a mobile device is 90ms, from SGSN to a mobile device is 101ms, and from GGSN to a mobile device is 114ms. We measure the PSNR of the received video at the mobile device. We assume that the video player in the mobile device can buffer 10 seconds worth of video and drop any frames that pass the assigned deadline.

Figure 10 shows PSNR when different approaches are used to send the test video for each of the three deployment options. The first bar shows the original compressed video. The second bar is the PSNR of a video sent by iProxy. iProxy decreases video bit rate when available network capacity is lower. Thus, it suffers no frame loss and its PSNR is only slightly lower than the original video. In the RNC case, its PSNR is 11% lower, and in the GGSN case it is 15% lower, showing that iProxy's performance suffers with network delay.

The third bar in Figure 10 shows the case where we keep the video bit rate as low as possible instead of adapting bit rate; this options can also prevent frame loss. However, lower video bit rate means lower quality video and lower PSNR; and even though there is available network capacity, it is left unused. Since this approach suffers no frame loss and uses a fixed bit rate, the PSNR is the same in the three scenarios, but is inferior to iProxy in all three cases. Furthermore, compared to iProxy, these schemes leave 20% of the bandwidth unused in the RNC case, 18% of the bandwidth unused in the SGSN case, and 15% of the bandwidth unused in GGSN case.

The fourth bar shows a conventional proxy sending high quality video directly to mobile devices. It results in a very high frame loss rate, and consequently has a poor PSNR. In addition, the video quality is affected much more drastically by network delay than iProxy. For example, PSNR drops 55% if we move the proxy from RNC to GGSN.

The last bar shows video quality when we stream the video directly from the video source. Without any help from proxy systems, the quality of video streaming is not acceptable with very low PSNR (i.e. 4.75dB). High delay and frame loss on the long path between video sources and mobile devices hurt PSNR significantly.

5.5 Summary

In sum, our evaluation shows the following:

- iProxy can improve cache hit rates from 20% to up to 1.6X for the two traces we studied, compared to conventional proxies using state-of-the-art cache replacement schemes. Using information-aware schemes is crucial. iProxy uses 2.4X less storage than a naive cache design.
- iProxy improves start up delays by 2-14s compared to a proxy that is not intelligent in selecting the right video version to serve to a client when a URL is requested. In some cases, iProxy can play video that a conventional design simply cannot play.
- iProxy's linear adapter improves bit rate by 16% compared to state-of-the-art MPEG DASH scheme. Our adaption scheme gracefully changes bit rate in response to changing network conditions. Buffering is minimized, if not virtually eliminated.

6. RELATED WORK

New video encoding schemes: In recent years, there have been proposals for new video encoding schemes that allow clients to play videos even in the face of partial errors in packets. For example, in [31], the authors argue that the whole video encoding process from DCT to 16-QAM encoding should be linear. They present SoftCast which replaces non-linear entropy coding by scaling up important DCT components to provide error protection in wireless networks. FlexCast [14] provides another way to encode a video grouping equally important bits of a video using "distortion grouping." Then, they modify Raptor codes [40] to do rateless video coding to protect more important groups. While these approaches could be used by iProxy, they all require a modification of the physical layer which makes deployment difficult. Our approach is targeted toward immediate deployment.

Video adaptation: [12] consider network conditions to send suitable versions of videos encoded using H.264/SVC [37] in a P2P VOD system. Their work can use network resources efficiently by picking the appropriate bit rate. However, the approach cannot provide linear bit rate.

Techniques such as HLS by Apple and Smooth Streaming [5] are very similar to MPEG DASH (they can be thought of as vendor-specific realizations of DASH). They share all of DASH's drawbacks discussed earlier, making it unsuitable for iProxy. In addition, Smooth Streaming and HLS require users to install a client side program.

In [35], the authors implement a cross-layer mechanism that optimizes TCP-Friendly Rate Control protocol to stream video on wireless multi-hop mesh networks. They use feedback from the receiver as an indicator to adjust sending rate. They control the sending rate in two ways: (a) at the physical layer, they change the modulation to employ suitable bit rates on wireless links, and (b) in the application layer, they change the quantization step in video encoding to achieve the right video bit rate. However, their solution needs to modify the receiver, where as ours doesn't. In addition, they also need to coordinate with the physical layer.

Vantrix [8] transcodes and adapts videos based on user devices and available bandwidth in real time. It results in efficient bandwidth usage and a better user experience. It also provides smart caching to store popular videos and save bandwidth on Internet backhaul links. However, Vantrix still use URLs to identify videos and may cache redundant videos. In addition, it transcodes videos from the original format instead of transcoding from frequency domain data, and as such, it is more resource intensive than our scheme (which skips the DCT process).

Video proxies: Others have proposed video proxies (e.g., MiddleMan [13, 42]) and some have focused on caching or prefectching strategies coupled with server-side scheduling that help offer better video experience and QoS [41, 34]. However, iProxy is the first to argue for an information-centric approach that achieves better caching of important information by aggregating multiple related URLs, and co-designs an effective video adaptation scheme into the cache. We design mechanisms that help directly improve quality metrics that matter for user engagement and abandonment. We also design and evaluate information-aware cache replacement policies.

7. CONCLUSION

This paper presented the design of a mobile video-centric cache named iProxy that simultaneously meets three key requirements: higher video hit rates at a lower overall cost, adaptability to channel diversity, and the ability to accommodate client diversity. iProxy leverages the recently-proposed notion of IBRs to identify and club together multiple variants of the same video into a single entity. It also uses a linear bit rate adapter that directly encodes raw in-cache video data at the optimal rate given instantaneous TCP feedback about network conditions. Both of these aspects of iProxy's design are rooted in the central observation that effective mobile video-centric cache design requires us to view caching from the higher level perspective of "information" as opposed to "data." Our evaluation shows that iProxy can improve hitrate, but we need to use novel information-centric replacement policies to achieve ideal benefits. We show that our linear encoder can adapt well to changes in bandwidth, and yield better bit rates and fewer buffering events than traditional approaches. We find that our scheme is able to deliver the optimal format to clients, minimizing start up delays.

Acknowledgements

We thank Vyas Sekar, Aaron Gember, and the anonymous MobiCom reviewers for their valuable comments on earlier versions of this paper. We also thank our shepherd Lakshmi Subramanian for his feedback. This research was supported in part by the National Science Foundation under awards CNS-1040757, CNS-0905134, and CNS-0746531.

8. REFERENCES

[1] android supported format. http://developer.android.com/guide/appendix/media-formats.html.

[2] Cisco visual networking index forecast, 2010. http://bit.ly/pIDtBX.

[3] Conviva. http://www.conviva.com.

[4] FFmpeg. http://ffmpeg.org/.

[5] Microsoft smooth streaming. `http://www.iis.net/downloads/microsoft/smooth-streaming`.

[6] Near-duplicate web video dataset. `http://vireo.cs.cityu.edu.hk/webvideo/`.

[7] The open source perceptual hash library. `http://phash.org`.

[8] Vantrix website. `http://www.vantrix.com/index.html`.

[9] video streaming server for ffmpeg. `http://ffmpeg.org/ffserver.html`.

[10] Vplayer official website. `https://vplayer.net/`.

[11] The ycbcr color space. `http://en.wikipedia.org/wiki/YCbCr`.

[12] O. Abboud, T. Zinner, K. Pussep, S. Al-Sabea, and R. Steinmetz. On the impact of quality adaptation in SVC-based P2P video-on-demand systems. In *MMSys*, New York, NY, USA, 2011.

[13] S. Acharya and B. Smith. *MiddleMan: A Video Caching Proxy Server*. NOSSDAV. 2000.

[14] S. Aditya and S. Katti. FlexCast: graceful wireless video streaming. In *MobiCom*, 2011.

[15] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, Jan. 1974.

[16] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transforms. *IEEE Trans. Computers*, 27(1):291–301, 1974.

[17] A. Anand, A. Akella, V. Sekar, and S. Seshan. A case for information-bound referencing. In *Hotnets*, 2010.

[18] A. Andoni. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *FOCS*, 2006:459—468, 2006.

[19] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. A quest for an internet video quality-of-experience metric. In *HotNets*, 2012.

[20] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakul. An active transcoding proxy to support mobile web access. In *In Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pages 118–123, 1998.

[21] J. S. Boreczky and L. A. Rowe. Comparison of video shot boundary detection techniques. 2670:170—179, 1996.

[22] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. A review of audio fingerprinting. *J. VLSI Signal Process. Syst.*, 2005.

[23] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *USITS*, 1997.

[24] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *SIGCOMM*, 2011.

[25] J. Erman, A. Gerber, K. K. Ramakrishnan, S. Sen, and O. Spatscheck. Over the top video: the gorilla in cellular networks. In *IMC*, 2011.

[26] A. Gember, A. Akella, J. Pang, A. Varshavsky, and R. Caceres. Obtaining in-context measurements of cellular network performance. In *IMC*, 2012.

[27] A. Gember, A. Anand, and A. Akella. A comparative study of handheld and non-handheld traffic in campus Wi-Fi networks. In *PAM*, 2011.

[28] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, San Francisco, CA, USA, 1999.

[29] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[30] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *IMC*, 2012.

[31] S. Jakubczak and D. Katabi. A cross-layer design for scalable mobile video. In *MobiCom*, 2011.

[32] S. Krishnan and R. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality using Quasi-Experimental Designs. In *IMC*, 2012.

[33] A. Kumar, A. Balachandran, V. Sekar, A. Akella, and S. Seshan. Infonames: An information-based naming scheme for multimedia content. Technical Report TR1677, UW-Madison, July 2010.

[34] J. Liu and B. Li. A qos-based joint scheduling and caching algorithm for multimedia objects. *World Wide Web*, 7(3):281–296, Sept. 2004.

[35] H. Luo, D. Wu, S. Ci, H. Sharif, and H. Tang. TFRC-Based Rate Control for Real-Time Video Streaming over Wireless Multi-Hop Mesh Networks. In *ICC*, Dresden, Germany, june 2009.

[36] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. 4th USENIX NSDI*, Cambridge, MA, Apr. 2007.

[37] 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*, Sept. 2007.

[38] S. Sen, J. Yoon, J. Hare, J. Ormont, and S. Banerjee. Can they hear me now?: a case for a client-assisted approach to monitoring wide-area wireless networks. In *IMC*, 2011.

[39] C. Serrano, B. Garriga, J. Velasco, J. Urbano, S. Tenorio, and M. Sierra. Latency in Broad-Band mobile networks. In *VTC*, 2009.

[40] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, June 2006.

[41] O. Verscheure, C. Venkatramani, P. Frossard, and L. Amini. Joint server scheduling and proxy caching for video delivery. In *WCCD*, pages 413–423, 2001.

[42] K.-L. Wu, P. S. Yu, and J. L. Wolf. Segment-based proxy caching of multimedia streams. In *WWW*, 2001.

[43] X. Wu, A. G. Hauptmann, and C.-W. Ngo. Practical elimination of near-duplicates from web video search. In *Multimedia*, 2007.