Effective Content-based Video Caching with Cache-friendly Encoding and Media-Aware Chunking

Sangwook Bae, Giyoung Nam, and KyoungSoo Park

Department of Electrical Engineering, KAIST
Daejeon, South Korea
{hoops, giyoung}@ndsl.kaist.edu, kyoungsoo@ee.kaist.ac.kr

ABSTRACT

Caching similar videos transparently in a network is a cost-effective solution that potentially reduces redundant data transfers. Recent study shows that network redundancy elimination (NRE) on the content level could produce high bandwidth savings in ISPs. However, we find that blindly employing existing NRE techniques to video contents could lead to suboptimal redundancy suppression rates. This is because (a) randomness in the video encoding process could produce completely different binaries even when they deal with seemingly identical video clips and (b) existing NRE chunking schemes incur high overheads since they do not utilize the underlying video format.

In this work, we present two novel schemes that help similar or aliased videos to be cached more effectively in the NRE system. First, we propose a deterministic video encoding scheme that preserves the unmodified original content even after editing or re-encoding. This would eliminate the sources of encoding randomness, allowing the NRE systems to detect the redundancy across similar videos. Second, we propose a lightweight video chunking scheme that exploits the underlying video structure. Our "sample-based" chunking scheme groups the logically-related frames into a chunk, and significantly reduces the size of NRE chunk indexes as well as chunking overheads. Our preliminary evaluation shows that the deterministic video encoding scheme helps greatly expose the redundancy across similar videos even after editing. Also, our sample-based chunking reduces the chunking overhead by a factor of 2.0 to 22.5 compared with popular NRE chunking schemes and reduces the index size by 27 times over various video contents.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems–Video

General Terms

Algorithm, Design, Performance

Keywords

Video Encoding, Network Redundancy Elimininination, Caching

MMSys'14, March 19–21, 2014, Singapore
Copyright 2014 ACM 978-1-4503-2705-3/14/03 ...$15.00
http://dx.doi.org/10.1145/2557642.2557657.

1. INTRODUCTION

Videos have already become the dominant traffic type in the Internet and are expected to be rapidly increasing in the near future [21]. To prepare for the growth, many Internet service providers (ISPs) heavily invest on efficient delivery of high-resolution videos in their fixed and cellular networks.

Caching in the ISP is an attractive approach to handling the fast-growing video traffic. It allows reducing redundant content transfers across the ISP networks by utilizing the network resources more efficiently. In addition, it improves the response time to the users, allowing more videos to be delivered in time. Since on-line video access is known to follow Zipf distribution [20, 40], even a small cache could significantly reduce redundant data transfers.

One of the most popular caching strategies is Web caching. While Web caching is a reasonable approach, it cannot suppress redundant transfers from aliased contents or those that have partial content overlap [37]. To maximize the bandwidth savings, one can run network redundancy elimination (NRE) on the content level. NRE divides the content into smaller chunks, and names each chunk using its content hash (e.g., SHA1 hash) to identify any duplicates even across different contents. A recent measurement study shows that even a simple NRE scheme can reduce as much as 59% of the entire cellular traffic, implying that many content transfers are redundant in practice [37].

When it comes to video contents, however, blindly applying NRE could produce suboptimal results. There are two reasons behind it. First, existing chunking methods for transparent NRE are inefficient for videos. Fixed-sized chunking could miss partial content overlap across different contents (e.g., videos that partially share the same clips). While variable-sized chunking like Rabin's fingerprinting [30] could suppress partial content overlap, it would not only increase the number of chunk indexes, but it also incurs significant chunking overheads. Second, even for the seemingly same video contents (e.g., the identical videos with the same resolution and frame rates), randomness in the video encoding process could produce completely different binaries. For example, the same movies served from different sites could have different binaries, which makes it impossible for transparent NRE systems to find the content-level redundancy.

In this paper, we propose two novel schemes that allow similar (or identical) videos to be more effectively cached in the NRE system. First, we present a deterministic video encoding scheme that produces identical binary for the portion of the overlapped contents in different videos. Our proposed encoding scheme is oblivious to editing or re-encoding, and produces the deterministic results as long as the target resolutions and frame rates are the same. For example, even if the encoding systems have a different

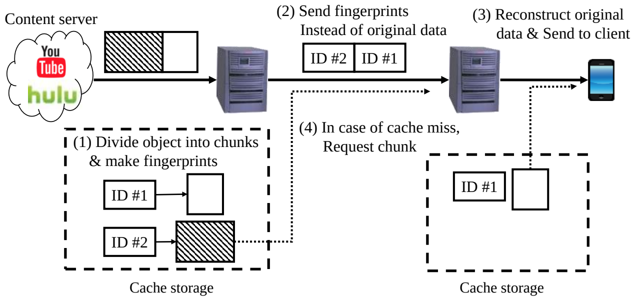


Figure 1: Network redundancy elimination

number of CPU cores, our encoding scheme ensures to have identical results as long as the input is the same. Second, we propose sample-based chunking for NRE systems that exploits the underlying video structure into chunking. A sample refers to a logical picture frame or an audio fragment in a video file. By utilizing the video format, we show that one can group semantically-related contents into one chunk, which results in a much smaller number of chunks for NRE with little chunking overheads compared with existing content-based chunking methods.

We implement our cache-friendly encoding library (called c264) by modifying x264 [36], the most popular open-source video encoding library, to preserve the redundancy from the original video regardless of editing or re-encoding. c264 allows NRE systems to easily detect and suppress the redundant content transfers for different versions of the same video. We find that c264 does not incur extra overhead in encoding the video while producing the similar video quality from x264. We also implement sample-based chunking and compare the performance of existing chunking schemes. Our evaluation shows that sample-based chunking reduces the number of chunks by up to 26.9 times compared with existing chunking methods, and detects the most of redundancy from similar videos regardless of the video types. Sample-based chunking is also faster than existing schemes by up to 22.5 times since it does not require expensive function evaluation that determines the chunk boundary. We also show that sample-based chunking can be extended to use group-of-pictures (GOP)[1] as a unit of a chunk, which further reduces the number of chunks at a slight cost of decreased redundancy detection.

The rest of the paper is organized as follows. In Section 2, we provide brief background of NRE systems and popular video formats and encoding schemes. We present some motivating examples in Section 3, and discuss our solutions in Section 4 and 5. We evaluate our new encoding and chunking schemes in Section 6, and show related work in Section 7. Finally, we conclude in Section 8.

2. BACKGROUND

In this section, we briefly provide the background on network redundancy elimination, the ISOBMFF video format, and the typical video encoding process.

2.1 Network Redundancy Elimination

An NRE system reduces the redundant data transfers in a bottlenecked network link. It typically consists of two middleboxes which sit at one end of the link as shown in Figure 1. A *sender* middlebox close to the traffic sender receives all the data that will

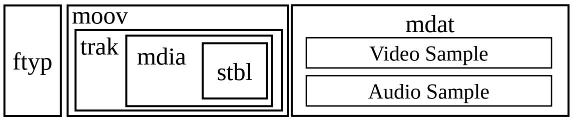


Figure 2: ISOBMFF

cross the link, and divides the content into smaller chunks. Instead of sending the chunk data, it first sends a stream of fingerprints (e.g., SHA-1 hashes of the chunks) to the other middlebox. If the other end has the actual content corresponding to the fingerprints, it reconstructs the original data and delivers it to the final destination. In case of a cache miss, it retrieves the chunk data from the sender middlebox, caches the data with its fingerprint as an index, and delivers it to the destination. Since these fingerprints are typically much smaller than the original chunks, if the cache hit rate is high, the NRE systems can bring significant bandwidth savings.

The process of dividing the content into smaller chunks is called chunking. Fixed-sized chunking divides the content into same-sized chunks. While fixed-sized chunking is simple, it could lead to suboptimal bandwidth savings since it may miss the redundancy from the contents that are slightly modified (e.g., adding one byte to a document would produce completely different chunks from the prior version). To overcome this problem, many forms of variable-sized chunking have been proposed. Rabin's fingerprinting computes a chunk fingerprint over the 32 bytes of sliding window for every byte and detects the chunk boundary if the value modulo 'p' equals to a pre-defined number (e.g., 0)[2]. Assuming the evaluated values are uniformly random, 'p' determines the average chunk size. Rabin's fingerprinting is widely used for deduplication since it is known to detect the redundancy even if the content is edited or updated [26, 29, 31, 33]. However, Rabin's fingerprinting incurs heavy computation overhead in the chunking process since it has to evaluate a function for each byte in the content. SAMPLEBYTE [17] overcomes this overhead by determining the chunk boundary with simple table lookups for each byte. That is, it first stores candidate byte values for a chunk boundary in a table, and scans each byte in the content to see if the byte is in the table. If a certain byte is in the table, then it simply generates a new chunk with that byte as the boundary of a chunk. SAMPLEBYTE drastically reduces the chunking overhead while it is known to produce a similar level of redundancy for network traffic [17].

However, one problem (especially relating to videos) is that both approaches do not consider the content format or types. While content-independent chunking has the benefit of generality, it would lead to suboptimal results for multimedia objects since it often makes completely-unrelated bytes into a chunk. Given that the video traffic is growing fast, video-aware chunking could produce more semantically-related chunks, which would reduce the number of chunks while improving the cache hit ratio in NRE systems. We focus on this aspect in this paper.

2.2 ISO Base Media File Format

We primarily focus on the ISO base media file format (ISOBMFF or MPEG-12 [16]) since it is one of the most widely-used video formats in progressive downloading or streaming [23]. As Figure 2 shows, an ISOBMFF file consists of boxes. Among many boxes, we look into *moov* and *mdat* boxes. The moov box stores the metadata of the media while the mdat box contains the actual video and audio data. A sample in a media format refers to an access unit

[1] A GOP is a set of successive pictures (typically between two successive I frames) in an encoding stream.

[2] This method is called MODP.

or an individual video frame in ISOBMFF. An mdat box can hold many video and audio samples. The moov box has the information of each media track for seeking, indexing, and decoding. More specifically, the *stbl* box in a moov box contains the offsets and sizes of the samples. We can extract the samples and boxes using this information in the moov box, which allows us to use its logical structure into chunking.

2.3 Video Encoding Process

We briefly review the video encoding process with H.264. H.264 is one of the most popular video encoding standards by Joint Video Team (or ITU's Video Coding Expert Group and MPEG) [6]. It is also known as MPEG-4 part 10 or Advanced Video Codec (AVC). x264 is a free software encoding library that implements the H.264 standard [36], which is widely used by video encoders or popular Web sites such as YouTube [14], Facebook [2], Vimeo [11], and Hulu [4].

H.264 video frames can be one of the three types: I, P and B frames. An I frame (or intra frame) serves as a reference frame that does not refer to other frames when it is encoded. In contrast, P and B frames (or inter frames) are typically encoded as the difference from other frames. When a video is edited or re-encoded, I frames in the output could be different from those in the original video. If I frames are chosen differently from the original version, all P and B frames that follow have to be encoded differently, which results in distinct binaries in the two videos even if they have content overlap. So, in order to preserve the redundancy in video editing, keeping and reusing the same I frames from the original content is important.

x264 implements the H.264 basic encoding flow by enabling parallel processing on multiple CPU cores. For parallel encoding, x264 employs three types of threads; one main thread, one or more lookahead threads and encoding threads. The main thread is in charge of the overall encoding procedure. When it receives decoded frames from a decoder, it places them into a lookahead input queue so that the lookahead threads can analyze them. Each lookahead thread decides the frame type based on its own inter-frame cost estimation, and reorders the frames to the encoding sequence order based on the frame reference order. When frame analysis is done, encoding threads encode those frames according to the frame type. The number of lookahead and encoding threads can be adjusted by the number of available CPU cores in the encoding system.

3. MOTIVATION

There are many routes to download popular videos from the Internet. People download movies, music videos, or TV shows from online video-on-demand (VoD) sites such as Netflix [8], Amazon [1], iTunes [5], and YouTube, or sometimes obtain them from file sharing sites [10]. The diversity of on-line video distribution sources naturally creates redundant transfers of the popular videos. According to recent studies, 27% of the videos returned from the top 25 popular queries at YouTube, Google Video [3], and Yahoo! Video [13] have either the same or almost identical content [38,39]. This implies that there is a high chance of reducing the redundant traffic by caching these identical contents.

To cache the same content transparently, ISPs could deploy NRE systems in their networks. However, even the videos with the same content often have different binaries, which could reduce the effectiveness of caching. The diversity on the binary level stems from different encoding environments, preset, and video containers (e.g, mp4, mov, avi, etc.) or could be due to re-encoding of the same content uploaded by the users. We often find no redundancy even between a full video and its video clips.

Encoders	1 CPU Core	8 CPU Cores	1 vs. 8 CPU Cores
VLC [35]	99.99%	19.94%	19.65%
Daumpot Encoder [22]	99.99%	2.45%	1.35%
HandBrake [34]	99.99%	99.99%	9.14%
Windows Movie Maker [28]	98.51%	98.91%	4.21%

Table 1: Redundancy between the two encoding outputs from the same video source. Parallel encoding often produces different binaries despite with the identical input.

To know more about the problem, we run a simple test. We download two music video files with the identical content (e.g., same length, resolution, frame rate, codec (H.264) in the same container format) from YouTube and Vimeo. We compare each video sample in these files, and confirm that they have completely different binaries. That is, the exactly same videos by "human eyes" could be encoded into a totally different binary.

One way to get around this problem is to cache the video based on its information bound reference (IBR) [32]. IBR extracts intrinsic features from each video, and compare them to identify the same content. IBR could identify the same videos despite with different binaries. However, one problem with IBR is that it could produce false positives, which recognizes different contents as the same video. What is worse is that this weakness can be exploited by an attacker that presents completely different videos as the same content.

The goal of our work is to identify possible causes for binary-level disparity for the same content, and to propose a new encoding scheme that encodes the same content into the same binary. Also, we suggest a new video chunking scheme that efficiently identifies the unit of caching by utilizing the underlying video format.

Re-encoding When content distributors obtain a video from a content provider, they often re-encode it to fit into their player options or to support different resolutions. However, this re-encoding process could produce the difference between original and new versions on the binary level.

One might think that using the same encoder with the same preset would solve this problem. However, we find that even with the same encoder, re-encoding sometimes produces different outputs on the binary level. Table 1 shows the redundancy across the two encoded outputs with the same video source as an input. That is, we re-encode a sample video (45-minutes long, HD resolution with 1280x720 pixels) twice, each time with the same encoder and the same preset, and calculate the redundancy between the two outputs by Rabin's fingerprinting with a 128-byte average chunk size. When we use a single CPU core for encoding, the two outputs are almost identical except for some metadata difference. However, when we encode with multiple CPU cores [3], the resulting binary is sometimes drastically different for each encoding as shown in VLC and Daumpot Encoder. In addition, we see the difference in the output binaries encoded with a single and multiple CPU cores, respectively. We suspect that the binary-level difference comes from

[3] We enabled Hyperthreading on a quad-core CPU machine.

Genre	Length (min)	Total # of I frames	Generated by scene-cut (%)
Action film [4]	123	2441	97.91
Animation [5]	10	141	88.65
TV show [6]	47	552	85.51
Sports (football)[7]	98	781	51.22
College lecture [8]	48	180	21.67
Video game [9]	64	488	10.86
Home video [10]	1.8	13	7.69
Adult	51	372	0.81

Table 2: % of I frames chosen by the x264 scene-cut function

randomness introduced by parallel encoding. For effective chunk-based caching, one needs to fix this problem.

Editing Partial video editing sometimes produces a completely different binary from the input even if the encoding parameters are same as in the source. For example, when you remove some frames in the front of a video while keeping the remaining frames intact, the starting I frame in the new version could be chosen differently from that in the original video. The difference in the starting frame affects the encoding process of the following frames, resulting in different B/P frames since the reference point is now different from that of the original source.

One workaround is to use the scene-cut function in x264. This function evaluates various metrics such as difference between nearby frames and a frame encoding/decoding cost to determine if a scene is being changed. Using this information, the encoder decides where to put an I frame to reflect a scene change. If we use the same scene-cut function to determine the location of the I frames both in the original and edited versions, we should be able to preserve the redundancy between the two files.

However, what we find is that the x264 scene-cut function works well only if a video includes frequent scene changes. Videos with little motion determine the locations of I frames at a fixed interval (e.g., by every 250 frames) rather than by the scene-cut function. When we edit these static videos, x264-based encoders would choose different locations as I frames even if the edited result is supposed to have significant content overlap from the original version. Table 2 shows this point as it compares the fraction of I frames selected by the scene-cut function for various videos of different genres. We observe the trend that videos with smaller scene changes are less affected by the scene-cut function in choosing the I frames. All this result shows is that we need to find a more dependable way to consistently select the I frames at video editing.

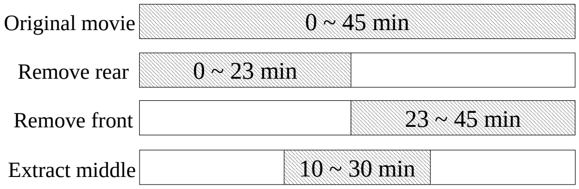
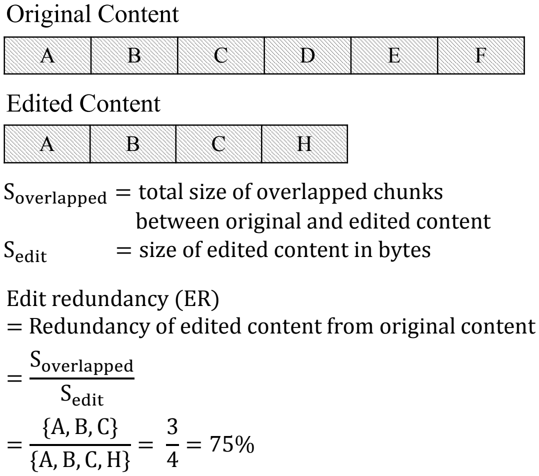


Figure 3: Video editing examples



$S_{\text{overlapped}}$ = total size of overlapped chunks between original and edited content

S_{edit} = size of edited content in bytes

Edit redundancy (ER)

= Redundancy of edited content from original content

$$= \frac{S_{\text{overlapped}}}{S_{\text{edit}}}$$

$$= \frac{\{A, B, C\}}{\{A, B, C, H\}} = \frac{3}{4} = 75\%$$

Figure 4: Definition of Edit Redundancy

Chunking Existing variable-sized chunking methods such as Rabin's fingerprinting may produce suboptimal redundancy suppression since they do not utilize the underlying media format. Even when two videos are composed of a set of identical samples, these samples could be laid out in a different order. For example, if video samples are interleaved by audio samples in a different order from those in the source, the chunk-level redundancy across the two versions would be small. Moreover, for maximum redundancy detection, a small chunk size is preferred in the existing chunking methods, which significantly increases the index management cost.

To know more about this, we compare the exposed redundancy by Rabin's fingerprinting in three cases of video editing: deleting the front half, deleting the rear half, and extracting the middle part of an original video (as shown in Figure 3). We edit the same video (45-minutes long) as in the previous section, and use MP4Box [7], which allows us to re-arrange or remove the samples in the original MP4 file. We define "edit redundancy" (ER) of original and edited versions as the ratio of the number of bytes in the shared chunks out of the total size of the edited file, as shown in Figure 4. For example, 100% ER means that the edited version is a complete subset of the original video and the chunking method exposes the maximum redundancy between the two.

Table 3 shows the ERs exposed by Rabin's fingerprinting over various average chunk sizes in three editing cases. Theoretically, we should see the ER as 100% in all cases since we simply cut some pieces from the original version. Rabin's fingerprinting exposes a good ER with a small average chunk size but the ER decreases rapidly as the average chunk size grows. We find that this is primarily because the audio samples are interleaved in the video samples in a different order from those in the original video. If a typical audio sample size is smaller than the average chunk size, more chunks would include a portion of both video and audio samples, which would reduce the ER in the chunk level.

To keep a high ER, one needs to use a small chunk size. However, a small chunk size would significantly increase the number of chunks, which incurs a high management cost of chunk indexes in

[4] Iron Man 3, the highest grossing movie of 2013 by IMDB [9]

[5] Big Buck Bunny. http://www.bigbuckbunny.org/

[6] Breaking Bad, the most popular TV series in 2013 by IMDB [9]

[7] Real Madrid vs. Chelsea football match (Aug. 8th, 2013)

[8] A lecture from MIT Open Courseware. http://youtu.be/zm2VP0kHl1M

[9] League of Legend all-star game in 2013, which drew 18 million unique viewers [24] http://youtu.be/crexqQ79g0o

[10] A popular home video at YouTube (with 62 million views) http://youtu.be/RP4abiHdQpc

Average chunk (bytes)	Original & Remove rear	Original & Remove front	Original & Extract middle
128	93.18%	93.17%	93.18%
256	88.33%	88.32%	88.33%
512	80.64%	80.77%	80.67%
1024	68.06%	67.96%	68.00%
2048	48.04%	47.92%	48.08%
4096	24.89%	25.22%	25.15%

Table 3: Edit redundancy exposed by Rabin's fingerprinting

Average chunk size (bytes)	# of chunks
128	9,584,983
256	4,785,126
512	2,391,610
1024	1,196,314
2048	598,018
4096	298,424

Table 4: # of generated chunks by Rabin's fingerprinting

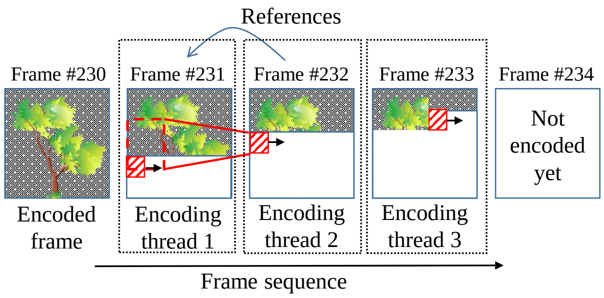


Figure 5: Example of thread synchronization. The gray area shows the portion that is already encoded, and the red-slashed boxes show the current encoding position. In a deterministic encoding mode, encoding thread 2 has to wait for the red-dashed box to be finished by encoding thread 1 (e.g., need to be synchronized) to correctly predict the inter-frame motion whereas in a non-deterministic mode, the search window for the motion prediction is shrunk not to block thread 2.

an NRE system. For example, an average chunk size of 128 bytes would produce 32 times more chunks than that of 4KB chunks. Besides the memory overhead, a large index size would increase a hash calculation cost as well. To give an example of the chunking overheads, we count the number of chunks detected by Rabin's fingerprinting for the 23 most popular HD videos in the YouTube chart[11] whose aggregate size is 1.3 GB. Table 4 shows the number of chunks over different average chunk sizes. The total number of samples in the dataset is 366,587 (video: 132,214, audio: 234,373), which is slightly larger than that of 4 KB chunks. This implies that any chunk size larger than 4 KB would produce a larger number of indexes than is required for the optimal redundancy.

In summary, we find a popular NRE chunking scheme like Rabin's fingerprinting is suboptimal for content-based video caching. First, a large chunk size exposes a smaller ER due to the non-deterministic interleaving of video and audio samples. Second, it could greatly increase the number of chunks compared with that of samples when the chunk size is small. Finally, Rabin's fingerprinting itself incurs high computation overheads since it has to evaluate a function for every byte in the video content. content types,

4. REDUNDANCY-PRESERVING ENCODER

In this section, we propose two schemes that preserve the redundancy between the original and edited video binaries. One is to remove the dependency on the encoding environment, and the other is to keep the positions of the I frames in the source video the same in the edited video. We find these two schemes work well in practice when we edit or re-encode a video.

We implement these two schemes in a cache-friendly library called c264 by slightly modifying the x264 video encoding library (ver.:2245). The implementation is simple since it requires only 200 lines of code out of 42K lines of the original x264 code.

4.1 Removing Environmental Dependency

We identify two sources of randomness in parallel video encoding with x264. First, we find that the non-deterministic encoding option in x264 often produces different binaries from the same input even when we use the same number of CPU cores for each encoding. Second, we identify a few x264 parameters that affect the encoding results depending on the number of CPU cores employed for encoding.

Removing randomness in the same environment For deterministic encoding output, we suggest that one should disable the non-deterministic encoding option in x264. This encoding option is used to improve the encoding quality and to maximize the parallelism in encoding in a multi-threaded mode. However, as the name implies, this option serves as the major source that produces inconsistent results despite with the same input since it dynamically changes several parameters while encoding.

There are two parameters that affect the encoding results when we use the non-deterministic encoding option. First, with this option on, the number of frames referenced by the lookahead threads is made to change over time, which affects the threads to decide the frame type. Due to this, a certain frame that is chosen be an I frame at one encoding could be made a P or B frame at another encoding since the number of referenced frames for determining the frame type is dynamic. Second, this option makes the inter-frame motion prediction range [12] variable over time. Variable motion prediction range would result in a different encoding binary at each encoding.

Disabling the option would make these two parameters fixed over time, which makes the consistent output from the same input as long as the encoding environment (e.g., the number of CPU cores) does not change. One might worry that disabling this option would affect the encoding speed or quality, but we find that it does not significantly slow down the performance as well as the encoding quality as shown in the evaluation section.

Removing dependency on the CPU core count The non-deterministic encoding option is not the only feature that injects randomness in parallel encoding. We find that the inter-frame motion prediction range and the number of lookahead threads depend on the number of CPU cores employed by the encoder. That is, these two parameters take on different values on a machine with a different number of CPU cores, which leads to difference in encoded output.

Here is how these two values affect the encoding output. x264 employs multiple encoding threads whose number is proportional to the number of CPU cores. As shown in Figure 5, multiple encoding threads would often require thread synchronization since some thread has to wait for inter-frame motion prediction range in the previous frame to be finished encoding to accurately predict the

[11] Sep. 23, 2013

[12] A search window box for motion prediction as shown in Figure 5

motion in its own frame. In the deterministic encoding mode, this range is fixed during encoding, but its value depends on the number of encoding threads. That is, it is set to a lower value as the number of encoding threads increases to reduce the chance of thread synchronization overhead. Due to this, encoding the same video with a different number of CPU cores would produce a distinct result even in the deterministic encoding mode.

A related problem happens with the number of lookahead threads. On a multicore system, x264 could have multiple lookahead threads, and they could analyze each frame in parallel. For example, one lookahead thread could look at the upper half of one picture (this portion is called slice) while another thread looks at the bottom half to figure out how previous frames would be used to encode each portion. So, two outputs would be different if they are encoded with a different number of lookahead threads.

To get around these problems, we fix the inter-frame motion prediction range (e.g., as the number used for eight cores) and set the number of lookahead threads to one regardless of the number of CPU cores. These two changes along with disabling the non-deterministic encoding option completely solves the problem of inconsistent output. One might suspect that these changes would degrade the encoding performance, but we find that the performance degradation is minimal as shown in the evaluation section. However, we do not claim our suggestion is free of any side effects, and we leave exploring and addressing them as our future work.

4.2 Keeping the IDR frame Position

Since I frames are referenced by P or B frames, having a different set of I frames in two logically identical streams of frames would make the content different on the binary level. That is, if we choose the location of I frames differently when editing a video (by cutting out some frames or concatenating with other frames), the two versions would have a completely different binary even if they are supposed to have significant overlap in the content.

To solve this problem, we keep the I frames in the original video to be preserved in the new version as long as the portion is not modified. To fulfill this requirement, we maintain the Instantaneous Decoder Refresh (IDR[13]) frame positions of the source video at the encoding process. That is, we obtain the IDR frame position from an stss box in each moov box in the source video since accessing to the stss box is the simplest way to get the information. In case the stss box does not exist, we obtain the information by parsing the mdat box.

5. VIDEO-AWARE CHUNKING

In this section, we present two chunking methods that exploit the underlying video structure into chunks. The goal of these approaches is to maximize the exposed redundancy across similar video contents while minimizing the chunking overhead. In addition, it should reduce the number of indexes to ease the index management cost.

5.1 Sample-based Chunking

Sample-based chunking makes each video or audio sample as a separate chunk. The key observation here is that the minimum granularity of a video file for the purpose of NRE is a sample. Since a video consists of a stream of samples, if two videos are said to have content overlap, they should have identical samples that represent the overlapped region. In this respect, a sample in a video file serves as the minimum unit that can be logically shared across multiple video files.

Algorithm 1 Sample-based Chunking Algorithm

```
procedure CHUNKING(VIDEO)
    //Parse metadata
    find moov box
    while there are remaining stbl boxes do:
        find the stbl box
        parse the location and the size of each sample
    end while

    //Divide samples into chunks
    $offset \leftarrow 0$
    while $offset$ < length of VIDEO do:
        if $offset$ is within an mdat box then
            make a chunk from the sample
            run SHA-1 and assign the chunk id
            $offset \leftarrow offset + sample\_size$
        else
            calculate Rabin's fingerprint
            determine whether to make a chunk or not
            $offset \leftarrow offset + 1$
        end if
    end while
end procedure
```

Employing a sample as the chunk unit presents a number of advantages. First, it exposes full redundancy that exist in two videos with any content overlap. This is because the content overlap among videos manifests as a form of a sample. Second, the chunking overhead of sample-based chunking is almost similar to that of fixed-sized chunking since finding chunk boundaries is as simple as offset calculation. In contrast, Rabin's fingerprinting has to evaluate a function for each byte to determine chunk boundaries, which incurs a much heavier overhead. Finally, it produces a much smaller number of indexes than that of existing variable-sized chunking schemes if we want a similar level of exposed redundancy. This is because existing chunking methods may miss the sample boundaries unless the average chunk size is set to a small value.

Algorithm 1 shows the pseudo code of sample-based chunking for NRE. First, it searches for the moov box by skipping other box types without parsing. It then extracts the offset and size information of every sample used in the file, which sits inside the stbl box in the moov box. Then, it scans the video and makes chunks out of the content. For an mdat box, it extracts each sample inside the box by its offset and size information. For other boxes, it applies Rabin's fingerprinting for getting the chunk boundaries. Then, it runs the SHA-1 hash function over each chunk data to obtain the ID for the chunk.

5.2 GOP-based Chunking

When we use sample-based chunking in practice, we notice one problem with it. While a large fraction of the video data is divided into a small number of chunks, it also produces a large number of small chunks. We find that this is primarily due to small audio samples whose number is typically much larger than that of video samples. These small chunks can degrade the overall the NRE system performance since it increases the number of I/Os as well as the content reconstruction cost in an NRE middlebox system.

One way to solve this problem is to convert only the video samples to chunks. That is, one can apply deduplication only on the video samples while sending other portion without NRE. This makes sense since audio does not take up much of the space in a video file, and the redundancy suppression ratio would not decrease too much.

[13] A special I frame defined by AVC [15] such that frames that come after an IDR frame cannot reference any frames before the IDR frame.

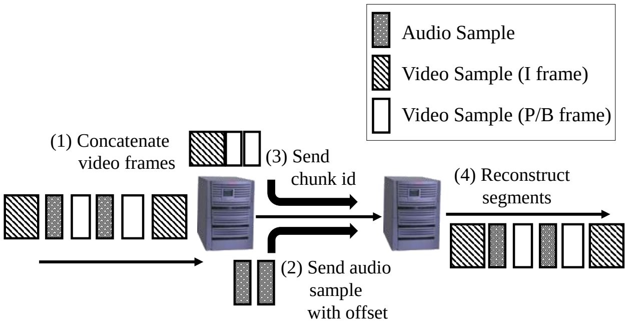


Figure 6: Overview of NRE system using GOP as a chunk

To verify this point, we analyze the popular YouTube videos used in section 3 again, and find that the audio data takes up only about 5 to 20 percent of the total size. In case of HD-quality videos, the audio portion is mostly less than 10 percent of the file size. Given that the demand for HD or higher-quality videos is rapidly increasing [21], skipping the audio data for NRE will be a good trade-off between the level of redundancy suppression and the NRE system performance. If we skip the audio samples, we can merge multiple video samples that are related into one chunk, significantly reducing the chunk overhead as well as the index size.

We define GOP-based chunking as the chunking scheme that makes all video samples between two consecutive IDR frames into one chunk. It skips the audio part for NRE and sends the data as is. Figure 6 shows how GOP-based chunking works with NRE middleboxes. First, the sender middlebox applies sample-based chunking to identify the chunk boundary, but expands each chunk by merging all samples between two IDR frames. In case of audio samples, it sends them with the offset information for content reconstruction later. Finally, the receiver middlebox reconstructs the original data and sends it to the client.

6. EVALUATION

In this section, we evaluate our encoding and chunking schemes. The goal of our evaluation is to show (a) c264 effectively preserves the redundancy across editing or re-encoding without performance degradation from x264 and (b) sample-based chunking reduces the chunking and indexing overheads while exposing full redundancy that exists between similar videos.

6.1 Effectiveness of Cache-Friendly Encoding

We show that our encoding scheme effectively preserves the redundancy from the original content even after editing/re-encoding and that the encoding overhead is comparable to that of existing schemes. We primarily compare our encoding library, c264, against the unmodified x264 library.

Exposed redundancy To show the benefit of c264, we encode all videos from various genres in Table 2 except for short videos (animations and home videos that are 1-minute long). We cut 5 minutes at a random position from each video and encode it with three encoders: Windows Movie Maker, x264-based and c264-based encoders. We re-encode the original videos as well, and compare the ER between the 5-minute clip and the re-encoded video. We use a machine with a quad core Intel CPU (i7 2600) with hyperthreading on and 16GB of physical memory. To prevent the effect from non-deterministic parallel encoding, we run Windows Movie Maker with a single CPU core while running the other two encoders by disabling the non-deterministic encoding option. We run the tests for five times, and calculate the average ER between the original video and the clips.

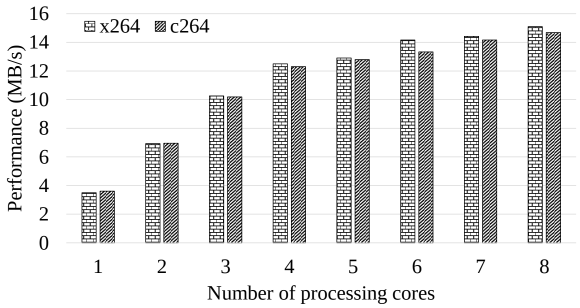


Figure 8: Encoding performance of c264 and original x264

Figure 7 compares the ER between the 5-minute clip and the re-encoded original video for each type. We calculate the redundancy using Rabin's fingerprinting with an average chunk size of 128 bytes. [14] We do not show the results for Windows Movie Maker since they all show less than 1% ER. This is mainly because Windows Movie Maker places IDR frames at a fixed interval, which makes the 5-minute clips start with a completely different IDR frame. From the figure, we find that c264 preserves the redundancy close to 100% from the original videos while x264 shows comparable results only for two genres. This shows that c264 is effective in preserving the redundancy regardless of the video type while the existing library could face a problem depending on the video content.

Performance of c264 We go on to measure the performance overhead of c264-based encoding. We choose a raw video from [12], which has 500 frames of full HD resolution (1920x1080) whose size is 1.6 GB. Figure 8 compares the encoding performance of x264 and c264 over various numbers of CPU cores. We find the tendency that the performance increases up to four CPU cores, and stabilizes more or less beyond that. We suspect this is because the encoding process is memory intensive, and hyperthreading does not improve the performance further. Overall, we find that the performance gap between the two schemes is within 3%, which implies that the extra overhead incurred by c264 is small. We also confirm that the encoding quality is almost the same. Both encoders produce the same peak signal-to-noise ratio (PSNR) as 33.298 for all 16 outputs, and the difference in the output file size is within 0.02%

6.2 Effectiveness of Sample-based Chunking

We evaluate the effectiveness of sample-based chunking in terms of execution time and ERs exposed by chunking schemes. We use a server-class machine with a hexacore Intel Xeon CPU (E5-2630) and 12 GB of physical memory.

Chunking execution time To evaluate the chunking overhead, we compare the execution time of sample-based chunking, Rabin's fingerprinting with MODP and SAMPLEBYTE [17]. We run these chunking algorithms on one of the most popular HD videos from YouTube [15] whose size is 1.1 GB. For fair comparison, we exclude the disk access overhead by pre-loading the content to memory before chunking.

[14] We avoid the interference by interleaved audio samples by encoding and checking only the video part.

[15] The video game shown in Table 2

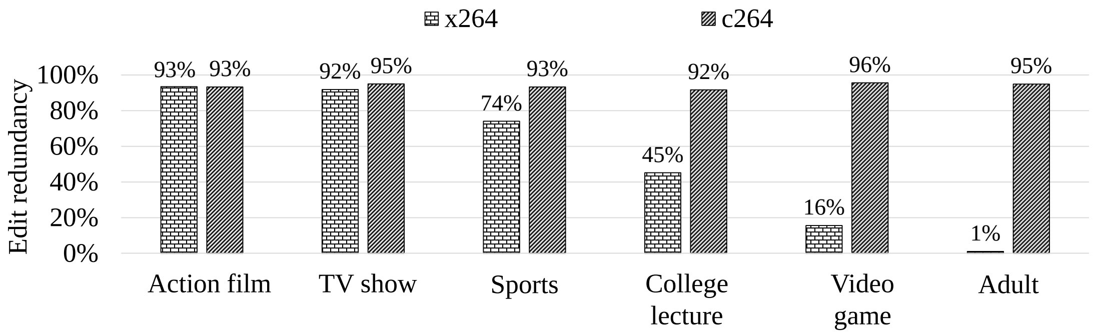
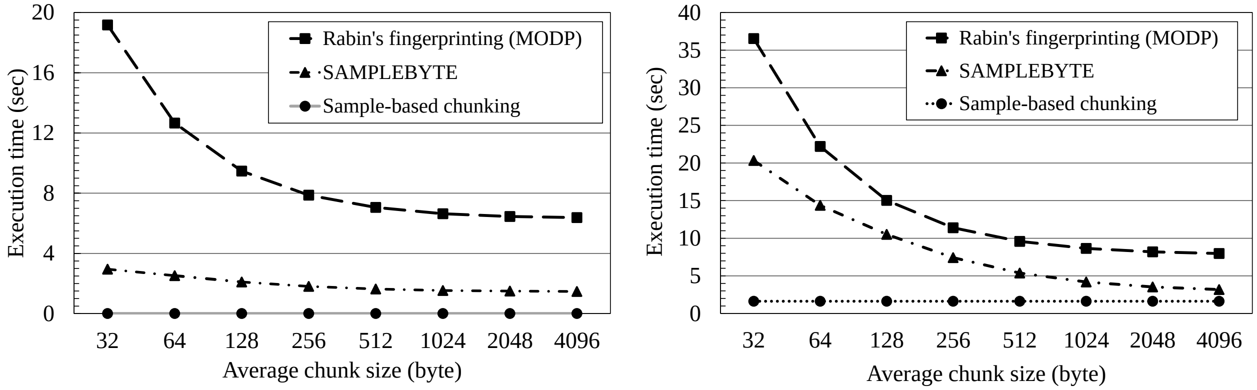


Figure 7: Edit redundancies between encoded and edited videos for various video types



(a) Chunking execution time without SHA-1 calculation

(b) Chunking execution time with SHA-1 calculation

Figure 9: Chunking execution time of a popular HD video (1.1 GB)

Figure 9(a) shows the results without SHA-1 calculation. Sample-based chunking takes 12.4 milliseconds, 510 and 1,585 times faster than Rabin's fingerprinting at 32B and 4 KB average chunk size, respectively. SAMPLEBYTE shows much better results than Rabin's fingerprinting but it incurs more overhead than sample-based chunking since it scans every byte for chunking. In contrast, sample-based chunking quickly finds the chunk boundaries by simple metadata lookup, which produces 117x to 236x speedups over SAMPLEBYTE.

Figure 9(b) compares the chunking times with SHA1 calculation. Sample-based chunking shows 4.9 to 22.5 (or 2.0 to 12.5) times performance improvement over Rabin's fingerprinting (and SAMPLEBYTE). The increased overhead by Rabin's fingerprinting and SAMPLEBYTE is primarily because the number of produced chunks is much larger than that of sample-based chunking, which exercises the fixed overhead of SHA-1 calculation per chunk.

Comparison of exposed redundancy To evaluate the exposed redundancy by sample-based chunking, we edit the video game video used in Section 3 by removing the front 30 minutes of the video with MP4Box. We then measure the ER between the two versions, and the total number of chunks generated by Rabin's fingerprinting and sample-based chunking.

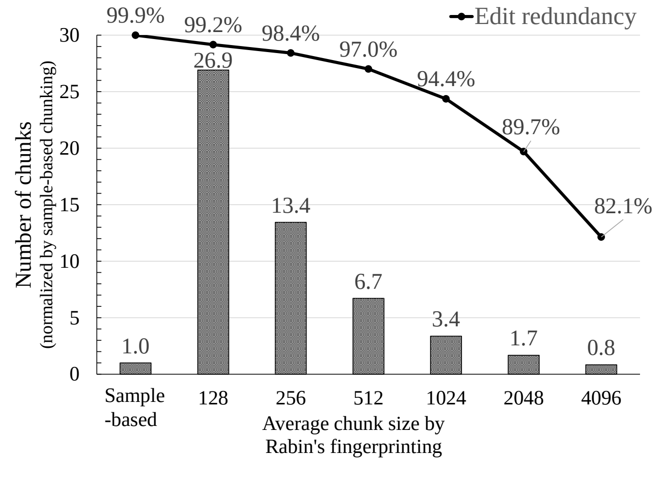


Figure 10: Edit redundancy comparison and total number of generated chunks according to chunking method

Figure 10 shows that sample-based chunking finds almost all redundancy (99.9%) that exists between the two versions. The reason why the ER does not reach 100% is because of the metadata difference in the moov box. Rabin's fingerprinting also detects the most of the redundancy at a small average chunk size, but the exposed redundancy decreases as the chunk size increases. With the 128 byte average chunk size, Rabin's fingerprinting shows the ER close to that of sample-based chunking, but it produces 26.9x more chunks, which incurs higher index management and content reconstruction costs. In addition, the small chunk size increases the chunking time by 9.6x than that of sample-based chunking. A 4 KB chunk size greatly reduces the number of chunks to even a smaller value (346,813) than that sample-based chunking (414,915), but it also reduces the ER by 17% as well as 4.9x larger chunking time

	Sample-based	GOP-based
Number of chunks	367,938	4,446
[Video chunks]	[132,214]	
[Audio chunks]	[234,373]	
Average chunk size (KB)	10.1	301.4

Table 5: Number of chunks and average chunk size of the 23 most popular YouTube videos

overhead. In summary, sample-based chunking exposes the most of the redundancy at a much smaller chunking overhead compared with Rabin's fingerprinting.

Effectiveness of GOP-based chunking We evaluate GOP-based chunking with the 23 most popular videos at YouTube used section 3. Table 5 compares the total number of chunks and the average chunk size by sample-based and GOP-based chunking. As shown in the Table, GOP-based chunking reduces the number of chunks by a factor of 82 than that of sample-based chunking, and the average chunk size is 29.8 times as large as that of sample-based chunking. One minor drawback of the current version of GOP-based chunking is that it skips the audio data for chunking, which takes up 9.3% of the total size.

We believe GOP-based chunking presents a great potential to reduce the chunk index management overhead as well as the chunk data I/O overhead in real video-aware NRE systems. However, we have not fully explored the trade-offs of GOP-based chunking yet, and leave other systems issues (e.g., real-time delay for GOP-based chunking, chunking of concatenated audio samples, overall system performance, etc.) to our future work.

7. RELATED WORKS

There have been a large body of studies that eliminate the redundant network traffic using content-based caching [17,19,25,29, 33,37,41]. One nice aspect of these systems is that they can be applied to any traffic as long as it has redundancy, which simplifies the system design and implementation. In our work, we focus on video-aware network traffic deduplication, a specialized form of NRE. For effective NRE of redundant video traffic, we suggest that the encoding algorithm should be free from the interference by parallelism, and that the chunking algorithm should reflect the video format into detecting the chunk boundaries. While our approach increases the engineering complexity, we believe it is worth the efforts given the performance benefits shown in this paper as well as the trend of rapidly increasing video traffic.

A recent work deals with caching the "perceptually" same videos of different encoding [32]. That is, it attempts to identify the videos whose content is indistinguishable by human eyes but that have distinct binaries due to encoding difference. They have built iProxy, a video caching proxy that saves the caching storage space by keeping only one copy of these perceptually same videos referenced by multiple URLs. To recognize the perceptually same videos, they use information reference bound (IBR), which quantifies a set of features in a multimedia object [18]. They coined VideoIBR which is a set of IBR values for a stream of chunks that constitute a whole video. For example, if a cache-missed video (fetched from the origin server) has an almost identical VideoIBR as one that is already stored, that video is simply linked to a previous copy. This scheme should be very useful in compressing the video cache storage given that videos are typically much larger than Web objects.

In comparison, we focus on NRE of similar video contents whose redundancy should be exposed as a portion of identical binaries. This approach guarantees safe operation by delivering the exact same copy to the client while VideoIBR could produce a possibility of identifying two different videos as the same one. Also, our scheme addresses partial content overlap between edited versions of the same video.

The are a few works [25, 27] that attempt to adjust the chunking algorithm to the content type for improved redundancy exposure. [27] employs specialized chunking for flash videos (FLV) and MP3 files and improves the data compression rate by up to 27% compared with that of Rabin's fingerprinting. [25] improves the NRE performance by adjusting the chunk size to the characteristics of the content. In comparison, our work focuses on encoding and chunking algorithms for ISOBMFF-aware NRE systems.

8. CONCLUSION

Eliminating redundant video traffic in a network is a cost-effective approach to handling fast-growing popularity of the video content. In this work, we have proposed a cache-friendly encoding scheme and two video-aware chunking algorithms. Our cache-friendly encoding scheme effectively exposes the underlying redundancy at the time of editing or re-encoding by removing the interference from parallel encoding. We find that our new encoding library, c264, preserves almost full redundancy between two versions at a low cost, compared with that of the x264 library. For video-aware chunking, we have proposed sample-based and GOP-based chunking. Sample-based chunking improves the chunking overhead by a factor of 2.0 to 22.5 compared with that of the best-performing chunking approach known so far while it exposes the redundancy almost perfectly. GOP-based chunking significantly reduces the chunk indexes by merging related samples into a chunk, which should be beneficial to reduce the NRE system overhead. We believe video-aware NRE is a promising area that needs to be further developed, and we hope our work will serve as the stepping stone towards the road.

9. ACKNOWLEDGMENTS

We appreciate many valuable comments on the earlier version of this paper from anonymous reviewers of MMsys 2013. This research was supported by the National Research Foundation of Korea (NRF) grant #2012R1A1A1015222.

10. REFERENCES

[1] Amazon Instant Video. http://www.amazon.com/aiv/.
[2] Facebook. https://www.facebook.com/.
[3] Google Video. http://video.google.com/.
[4] Hulu. http://www.hulu.com/.
[5] iTunes. https://itunes.apple.com.
[6] Joint Video Team. http://www.itu.int/en/ITU-T/studygroups/com16/video/Pages/jvt.aspx.
[7] MP4Box. http://gpac.wp.mines-telecom.fr/mp4box/.
[8] Neflix. http://www.netflix.com/.
[9] The Internet Movie Database. http://imdb.com/.
[10] Top 15 Most Popular File Sharing Websites. http://www.ebizmba.com/articles/file-sharing-websites.
[11] Vimeo. https://vimeo.com/.
[12] xiph Open Source Community. http://www.xiph.org/.
[13] Yahoo! Video. http://screen.yahoo.com/.
[14] Youtube. https://www.youtube.com.
[15] ISO/IEC 14496-10:2012, Information technology - Coding of audiovisual objects - Part 10: Advanced Video Coding. Technical report, 2012.

[16] ISO/IEC 14496-12:2012, Information technology - Coding of audiovisual objects - Part 12: ISO base media file format. Technical report, 2012.

[17] B. Agarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese. EndRE: An end-system redundancy elimination service for enterprises. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NDSI)*, 2010.

[18] A. Anand, A. Akella, V. Sekar, and S. Seshan. A case for information-bound referencing. In *Proceedings of the ACM International Workshop on Hot Topics in Networks (HotNets)*, 2010.

[19] A. Anand, V. Sekar, and A. Akella. SmartRE: an architecture for coordinated network-wide redundancy elimination. In *ACM SIGCOMM Computer Communication Review (CCR)*, volume 39, pages 87–98, 2009.

[20] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2007.

[21] Cisco, Inc. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012-2017, 2013.

[22] Daum, Inc. Daumpot Encoder. `http://tvpot.daum.net/application/PotEncoder.do`.

[23] J. Erman, A. Gerber, K. Ramakrishnan, S. Sen, and O. Spatscheck. Over the top video: the gorilla in cellular networks. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2011.

[24] Forbes. League of Legends Drew 18 Million Viewers For Its All-Star Game. `http://www.forbes.com/sites/insertcoin/2013/07/11/league-of-legends-drew-18-million-viewers-for-its-all-star-game/`.

[25] E. Halepovic, C. Williamson, and M. Ghaderi. Enhancing redundant network traffic elimination. *Computer Networks*, 56:792–809, 2012.

[26] S. Ihm, K. Park, and V. Pai. Wide-area network acceleration for the developing world. In *Proceedings of the USENIX Annual Technical Conference (USENIX)*, 2010.

[27] C. Liu, Y. Lu, C. Shi, G. Lu, D. Du, and D. WANG. ADMAD: Application-Driven Metadata Aware De-duplication Archival Storage System. In *Proceedings of the IEEE International Workshop on Storage Architecture and Parallel I/O (SNAPI)*, 2008.

[28] Microsoft, Inc. Windows Movie Maker. `http://windows.microsoft.com/en-us/windows-live/movie-maker`.

[29] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

[30] M. Rabin. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[31] S. Rhea, K. Liang, and E. Brewer. Value-based web caching. In *Proceedings of the international conference on World Wide Web (WWW)*, 2003.

[32] S. Shen and A. Akella. An information-aware qoe-centric mobile video cache. In *Proceedings of the ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2013.

[33] N. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *ACM SIGCOMM Computer Communication Review (CCR)*, volume 30, pages 87–95, 2000.

[34] The HandBrake Team. HandBrake. `http://www.handbrake.fr/`.

[35] VideoLAN. VLC Player. `http://www.videolan.org/vlc/`.

[36] VideoLAN. x264 specification. `http://www.videolan.org/developers/x264.html`.

[37] S. Woo, E. Jeong, S. Park, J. Lee, S. Ihm, and K. Park. Comparison of Caching Strategies in Modern Cellular Backhaul Networks. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2013.

[38] X. Wu, A. Hauptmann, and C. Ngo. Practical elimination of near-duplicates from web video search. In *Proceedings of the ACM international conference on Multimedia (MM)*, 2007.

[39] X. Wu, C. Ngo, and A. Hauptmann. CC WEB VIDEO: Near-Duplicate Web Video Dataset. `http://vireo.cs.cityu.edu.hk/webvideo/`.

[40] H. Yu, D. Zheng, B. Zhao, and W. Zheng. Understanding User Behavior in Large-scale Video-on-demand Systems. In *Proceedings of the ACM SIGOPS European Conference on Computer Systems (EuroSys)*, 2006.

[41] E. Zohar, I. Cidon, and O. Mokryn. The power of prediction: Cloud bandwidth and cost reduction. In *ACM SIGCOMM Computer Communication Review (CCR)*, volume 41, pages 86–97, 2011.