RATE EFFICIENT REMOTE VIDEO FILE SYNCHRONIZATION

Hao Zhang, Chuohao Yeo, and Kannan Ramchandran

Department of EECS, University of California, Berkeley
Berkeley, CA 94720, USA

ABSTRACT

Video file synchronization between remote users is an important task in many applications. Re-transmission of a video that has been only slightly modified is expensive, wasteful and avoidable. We propose a scheme that automatically detects and sends only modified content according to some user defined distortion constraint to enable rate savings under a wide range of video edits. Through the use of a low-rate hierarchical hashing scheme, we can detect modifications with some spatial granularity. We also apply distributed source coding techniques to exploit correlation between remote copies for a further rate rebate. Experimental results show that the proposed approach achieves up to 7× rate reduction when compared to re-transmitting.

Index Terms— VSYNC, rsync, video file synchronization, video hash, video coding

1. INTRODUCTION

File synchronization is an important and common tool for making incremental updates in applications like backing up or mirroring to remote sites. Tools such as rsync [1] perform *exact* bit-stream synchronization of files and directories between remote locations while minimizing data transfer by essentially sending only the difference. When attempting to synchronize video content, content-agnostic approaches such as applying rsync to compressed video files fail because a minor modification could result in a completely different bit-stream. Furthermore, they do not allow synchronization to within a specified *distortion*.

Video file synchronization, which we define as maintaining similar video content up to distortion across remote sites, is also an important tool. Consider the following motivating examples. A worker at a company headquarters prepares a demo video and sends a compressed version to other company offices around the world. He then does some video editing and wants to update everybody's copy of the video. Re-transmitting the entire video file with only small changes is expensive and unnecessary, especially in situations where the video file is large and communication cost is high. Another example is uploading (downloading) video content to (from) a video file sharing site such as YouTube. When a user uploads (downloads) a video similar to one that is already present, it would be extremely helpful if only the modified content is uploaded (downloaded), which avoids transferring the entire video and hence substantially reduces Internet traffic. In another scenario, it may be desirable to allow mobile devices to always have access to the most recent version of a video located on a remote server, but the local copy could have different resolution and/or bit-rate due to the device characteristics and limitations. Minimizing transmission rate would be important due to limited bandwidth and energy.

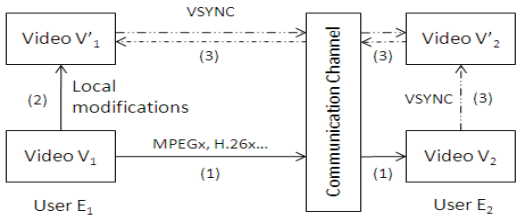


Fig. 1. A typical video file synchronization setup. Initially, V_2 is some compressed and/or down-sampled version of V_1; V_1 is later modified to $V_1^{'}$. We seek to synchronize to this modification by updating V_2 to $V_2^{'}$ at pre-defined quality using minimal data transfer between E_1 and E_2.

In this paper, we consider the following video file synchronization problem shown in Figure 1. Initially, two users E_1 and E_2 at remote locations have videos V_1 and V_2 respectively, which are of the same content. Step (1) in Figure 1 indicates that V_2 may not be exactly the same as V_1, e.g. it may be a compressed and/or spatially down-sampled version of V_1 that is encoded and transmitted to E_2. Step (2) denotes some video editing, where user E_1 updates V_1 to $V_1^{'}$. Now, user E_2 wants to keep V_2 synchronized to $V_1^{'}$ to get $V_2^{'}$ at some pre-defined quality through step (3), ideally with minimal data transfer. The same notations in Figure 1 will be used throughout the paper unless stated otherwise.

We have previously proposed a video file synchronization protocol, VSYNC, to address the above problem [2], i.e. step (3) in Figure 1. In this work, we improve upon the hierarchical hashing scheme which converts the high-level content information to a low-level hash stream that is more amenable to coding tools. Specifically, the hash structure is now designed to spatially localize modifications. In addition, we investi-

gate the application of distributed video coding (DVC) techniques [3, 4] to exploit the correlation between frames from $V_1^{'}$ and V_2 to gain further rate savings. Experimental results show that the proposed modifications improve performance and outperform rsync and re-transmitting using H.264.

2. BACKGROUND

When an update request is initiated using rsync, it compares a weak hash, and if necessary a strong hash, of chunks of two files to find chunks in common between them, and send only the chunks that are different as well as necessary assembly instructions [1]. The weak hash is to quickly rule out different chunks, while the strong hash is applied to confirm matches. A weak/strong hash division much inspired by rsync is used in VSYNC, but the hashes operate in a content-aware and distortion tolerant manner due to the nature of video files.

There are various image and video hashing techniques in the literature designed for tamper detection [5, 6, 7]. Of particular interest is a scheme proposed by Lin et al. [7], which encode hashes of the original image as syndromes of a suitable low-density parity check (LDPC) code. On the other hand, VSYNC is a systematic multimedia protocol that is designed to efficiently synchronize video files rather than detect image tampering. Furthermore, the thresholds and hash rates in the image tampering literature are typically picked empirically based on training data. In contrast, VSYNC [2] uses a binary hashing scheme with a well-understood property that relates hamming distance between hashes constructed from binarized random projections to distortion between video frames [8], thus allowing for a principled approach to making such system choices.

3. SYSTEM DESIGN

3.1. System Overview

The design goal of VSYNC is to synchronize video files remotely with efficient bit-rate utilization. In this work, we design a hashing scheme such that matches are verified on a per-macroblock (per-MB) basis (across multiple frames) instead of on a per-frame basis [2], to provide additional spatial granularity. Second, we apply DVC [3, 4] to efficiently update frames that are well-correlated but do not satisfy the desired criteria, and make a novel use of the hash check to estimate correlation between remote copies of the frames. These proposals help save substantial rate compared to rsync and H.264.

We define a Group of Frames (GOF) as a chunk of M frames. Each frame of size N_1 x N_2 is divided into q x q macroblocks (MB). When an update request is initiated, VSYNC checks by comparing a weak hash of each overlapping GOF of $V_1^{'}$ and non-overlapping GOF of V_2 to cheaply rule out mis-matches. If necessary, V_2 should first be resized to the same spatial dimensions of $V_1^{'}$ before computing the hashes. VSYNC then verifies the matches by checking a strong hash of each MB across the corresponding GOFs, which helps to detect localized edits and thus avoids re-transmission of unmodified blocks. MBs that do not pass the hash check are encoded and transmitted using a suitable codec such as H.264. Since video edits tend to occur in spatially and temporally contiguous regions, such MBs can be efficiently encoded. For MBs that pass the strong hash, the hamming distance between the underlying hashes is used to determine if they are of a desired quality. If not, the hamming distance is used to estimate the correlation between MBs from V_1' and V_2, which is in turn used for rate allocation in updating the MBs with a DVC approach. This allows further rate reductions by exploiting the presence of correlated frames even if they do not satisfy the desired distortion. E_1 then sends over necessary assembly instructions for E_2 to arrange the frames and blocks in the correct order. Algorithm 1 summarizes the protocol.

Algorithm 1 VSYNC Protocol

1: Upon update request, E_2 computes a weak hash for every non-overlapping GOFs $i = 1, \ldots, m$ and the corresponding LDPC parity bits of the strong hashes for the MBs within each GOF, and sends them to E_1.
2: E_1 computes the weak hashes for every overlapping GOF $j = 1, 2, ..., n$ and does the following loop (steps 3-8).
3: **for** $i = 1$ to m, $j = 1$ to n **do**
4: Check if the weak hashes are within threshold T_w [2]. If not, continue; otherwise:
5: **for** $b = 1$ to B **do**
6: Apply BP decoding to the LDPC parity bits of the strong hash for MB b. If successful, check the hamming distance to determine and send to E_2, if necessary, the number of additional updating syndrome bits of video encoding to ensure desired quality, using a DVC approach described in Section 3.3; if the check fails, send the MB using H.264 or other suitable codecs.
7: **end for**
8: **end for**
9: E_1 sends frames that are not of any GOF match in $V_1^{'}$ and necessary assembly instructions to E_2 to reconstruct $V_2^{'}$.

3.2. Hierarchical Hashing for Edit Localization

The following hash generation process is adopted. We first generate KB random matrices $P_{k,b} \in \mathbb{R}^{q \times q}, k \in \{1, \ldots, K\}$, $b \in \{1, \ldots, B\}$ whose entries are i.i.d uniform random variables, where B is the total number of MBs within each frame and K is the number of projections per MB. For each MB b in frame l of GOF i denoted as $v_{il,b}$, K Frobenius inner products $\mathbf{Tr}(P_{k,b}^T v_{il,b})$ are each quantized into one bit depending on its sign. This is repeated for all frames within GOF i, i.e. $l = 1, \ldots, M$, where M is the number of frames within GOF i, to get the binary string s_i. This spatial separation helps to localize edits, e.g. adding a logo or removing a certain object at some fixed location across the GOF. As will be shown in our results, this hashing

scheme successfully avoids sending entire frames when only minor modification is made on a small portion of the frames.

The projection bits of similar video blocks are expected to have low hamming distance [8], and their relationship can be modeled using a Binary Symmetric Channel (BSC). For some distortion Q (in PSNR), the corresponding crossover probability can be derived as $p(Q) = \frac{2}{\pi}\sin^{-1}\frac{255}{2\sigma 10^{Q/20}}$ [2], where σ^2 is the pixel intensity variance of the video. We denote Q_d the desired video quality, and $Q_h < Q_d$ the minimum required video quality. We also define $p_d = p(Q_d)$ and $p_h = p(Q_h)$.

Based on these projections, we design the weak hash to check on a GOF basis to cheaply rule out GOFs that are far away from each other, by choosing to send at random a fraction r of the total projection bits from s_i. This selection saves bit-rate while also allowing it to uniformly capture the possible editing done in the GOF. We can then use a threshold $T_w = rMKBp_h$ for the hamming distance check between the weak hashes [8] and only corresponding GOFs that satisfy video quality Q_h will pass the weak hash check.

On the other hand, the strong hash is designed to confirm if indeed each group of spatially co-located MBs across the GOFs satisfy the video quality Q_h, and if so, how much new information needs to be sent for the update to satisfy the desired quality Q_d. In particular, the K projection bits for each $q \times q$ MB b across the entire GOF are strung together to form its hash bits $s_{i,b}$. To reduce the rate of the strong hash, we apply the parity check matrix of a LDPC code to the hash bits to obtain and transmit only the parity bits [7, 9], using a coding rate such that its threshold is about p_h. User E_1 would then use a BP decoder [10] to decode the LDPC parity bits using its corresponding projection bits as side-information; decoding will be successful if the two hash sequences satisfy the assumed correlation model. While the use of the LDPC parity bits results in rate savings, BP decoding also requires more computations, and thus we only apply the strong hash check on GOFs that pass the weak hash.

3.3. Rate Efficient Update

The weak-strong hash check described in Section 3.2 helps E_1 to decide the necessary update information to be sent to E_2 for synchronization. These updates include two possibilities. First, frames from V_1' that are not part of any GOF match after the weak hash check are grouped together and encoded using a suitable codec at the desired quality Q_d. Second, for each MB within the GOFs that passed the weak hash, BP decoding of the strong hash is performed.

Decoding failure indicates they do not satisfy the minimum quality Q_h, and the MBs across all frames within that GOF are coded with a standard video codec; otherwise, a DVC approach [4] is applied to update, if necessary, the MBs, to ensure a quality of Q_d. Denote a pair of corresponding MBs in $V_1^{'}$ and V_2 as X and Y respectively; user E_1 would like to transmit X to E_2, which has Y as side-information. A 2-D DCT is applied on X, and the transform coefficients are quantized and zig-zag scanned before syndrome encoding [4]. A total of 14 correlation classes are trained offline and the appropriate correlation class for X and Y is determined by the hamming distance between hashes. This inferred correlation is then used to determine the syndrome rate for each DCT coefficient. E_2 then decodes the received syndrome bits of the coefficients using Y as side-information, unquantizes the coefficients, undoes the zig-zag scanning, and applies the 2-D IDCT to reconstruct the block.

4. EXPERIMENTAL RESULTS

4.1. Setup

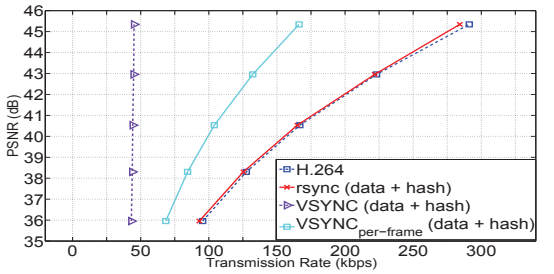
The test videos used in this work are the "News (QCIF)" (176x144, 25fps, 12s), "News (CIF)" (352x288, 25fps, 12s), "Foreman (QCIF)" (176x144, 25fps, 12s) and "Foreman" (CIF) (352x288, 25fps, 12s). For each pair of videos (QCIF and CIF), two video editing cases are considered:

- **Case** 1**.** V_1 is the raw QCIF and is compressed with H.264 using Group of Pictures (GOP) size 15 to obtain V_2. To form $V_1^{'}$, swap the 2$^{\text{nd}}$ and the 3$^{\text{rd}}$ GOP and add a logo of size 52 by 32 at the upper left corner across the first 150 frames. The target synchronization quality of $V_2^{'}$ is the same as that of V_2. The choice of parameters is: GOF size $M = 20$, MB size $q = 16$, number of projections per MB $K = 64$, and down-sampling fraction $r = 0.01$.
- **Case** 2**.** V_1 is the raw CIF, and is down-sampled to QCIF and compressed with H.264 using GOP size 10 to obtain V_2. To form $V_1^{'}$, the last 20 frames of V_1 are deleted. The goal is to up-sample V_2 to V_1's resolution and update its content to $V_1^{'}$. The choice of parameters is: $M = 10$, $q = 32$, $K = 128$ and $r = 0.01$.

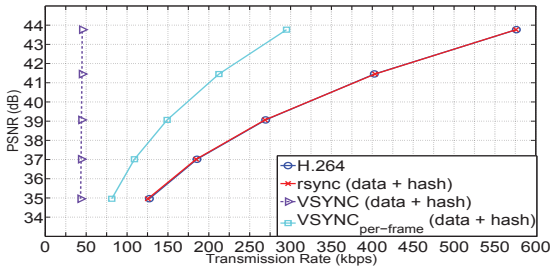
We compare the following methods: (1) rsync: compress $V_1^{'}$ to $V_2^{'}$ using H.264 with GOP size 15 at the targeted PSNR and run rsync to update V_2 to $V_2^{'}$; (2) H.264: compress $V_1^{'}$ to $V_2^{'}$ using H.264 at the targeted PSNR and directly send it over; (3) VSYNC (per-frame) [2]: run the algorithm between $V_1^{'}$ and V_2 to update V_2 to $V_2^{'}$; and (4) VSYNC: run the proposed scheme between $V_1^{'}$ and V_2. It is worth noting that the strong hash rate and threshold for VSYNC (per-frame) and VSYNC are different. The latter allows a larger distortion which increases the hash overhead but helps reduce the rate to update the frames especially when the contents are highly correlated.

4.2. Results

Figures 2 and 3 shows the rate-distortion (RD) curve for cases 1 and 2 respectively, where distortion is measured in PSNR between videos $V_1^{'}$ and $V_2^{'}$ and rate (in kbps) is the rate of information exchanged in both directions needed to update the videos. In case 1, the results show that even when only a small fraction of the frames are edited, rsync and H.264 end up sending the entire updated video, which is unnecessary.

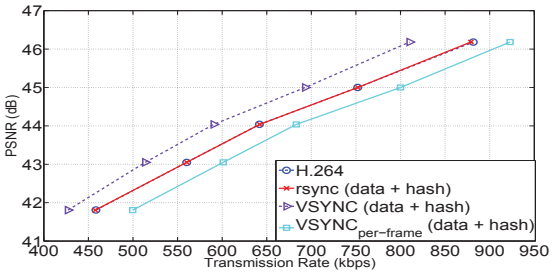


(a)

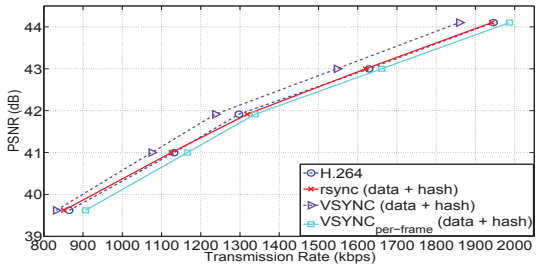


(b)

Fig. 2. RD curve under case 1. (a) "News" (b) "Foreman".



(a)



(b)

Fig. 3. RD curve under case 2. (a) "News" (b) "Foreman".

VSYNC is able to detect the text banner and only transmit the corresponding content. Note that VSYNC (per-frame) [2] has a performance in between the VSYNC and the other two approaches. In fact, it transmitted the first half of the video that has a text banner added to it, while keeping the second half which were not edited at all. It is worth noting that PRISM is not utilized in case 1, since the video quality at E_2 has already satisfied the constraints. In case 2, V_2 after upsampling is still correlated with V_1, and only the missing details should be transmitted. Rsync is again unable to exploit the correlations between the videos, and thus is as expensive as re-transmitting the entire video using H.264. VSYNC (per-frame) [2] also fails to utilize the correlation, since V_2 after up-sampling will no longer satisfy the desired quality constraint. The proposed VSYNC offers rate savings even in this case. We do point out one caveat: the rate savings also depend on how users edit the video and can be made arbitrarily large compared to other approaches, e.g. one can edit an infinitely long video such that a logo is added to every frame, causing rsync and VSYNC (per-frame) to retransmit the entire file.

5. CONCLUSIONS AND FUTURE WORK

We extend the video file synchronization system presented in [2] by adding a new hierarchical hash structure to localize various edit and applying a rate efficient DVC scheme to update contents. Our experimental results show an improvement in rate savings under the considered test cases. Future research directions include (1) taking into account temporal correlation to build more rate efficient hashes; (2) develop more rate-efficient DVC algorithms; and (3) optimize the trade off between rate for hash and that for the updates.

6. REFERENCES

[1] A. Tridgell and P. Mackerras, "The rsync algorithm," `http://rsync.samba.org/`, Nov 1998.

[2] H. Zhang, C. Yeo, and K. Ramchandran, "VSYNC — A Novel Video File Synchronization Protocol," *to appear in Proc. ACM International Conference on Multimedia (ACM MM)*, 2008.

[3] B. Girod, A. Aaron, S. Rane, and D. Rebollo-Monedero, "Distributed video coding," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 71–83, 2005.

[4] R. Puri, A. Majumdar, and K. Ramchandran, "PRISM: A Video Coding Paradigm With Motion Estimation at the Decoder," *IEEE Transactions on Image Processing*, vol. 16, no. 10, pp. 2436–2448, 2007.

[5] J. Fridrich and M. Goljan, "Robust hash functions for digital watermarking," *International Conference on Information Technology: Coding and Computing*, pp. 178–183, 2000.

[6] B. Coskun and B. Sankur, "Robust video hash extraction," *IEEE 12th Proceedings on Signal Processing and Communications Applications Conference*, pp. 292–295, 2004.

[7] Y.C. Lin, D. Varodayan, and B. Girod, "Image Authentication and Tampering Localization using Distributed Source Coding," *IEEE 9th Workshop on Multimedia Signal Processing*, pp. 393–396, 2007.

[8] C. Yeo, P. Ahammad, and K. Ramchandran, "Rate-efficient Visual Correspondences using Random Projections," *to appear in IEEE International Conference on Image Processing*, 2008.

[9] S.C. Draper, A. Khisti, E. Martinian, A. Vetro, and J.S. Yedidia, "Using Distributed Source Coding to Secure Fingerprint Biometrics," *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 2, 2007.

[10] T.J. Richardson and R.L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, 2001.