A real-time video watermarking system with buffer sharing for video-on-demand service

Ju Wang\textsuperscript{a,*}, Jonathan C.L. Liu\textsuperscript{b,*}, Mbonisi Masilela\textsuperscript{a}

\textsuperscript{a}Computer Science, Virginia Commonwealth University, Richmond, VA, USA
\textsuperscript{b}CISE Department, University of Florida, Gainesville, FL, USA

Abstract

Real-time watermarking for streaming video (such as VOD service) requires significant amounts of computing resources. To address this issue, we present a scalable watermarking scheme integrated in a parallel MPEG-2 engine. A content-based block selection algorithm is proposed to efficiently embed the pseudo-random watermark signatures into DCT blocks. Our watermark scheme also provides a robust way to synchronize the watermarked video to the original source at detectors and is very resilient against cumulative and temporal attack.

We optimize the parallel watermark engine to achieve real-time watermarking performance. We found that the system throughput could suffer significant degradation when processing high-level MPEG-2 video (such as HDTV) due to inefficient management of memory space. Therefore, we investigated an efficient buffer management scheme consisting of two methods: First we reduced the transmission buffer in slave nodes by frames sharing between frames in the Group-of-Picture (GOP) level. Then we further reduce the buffer space by a dynamic on-demand allocation on the slave side. By solving the memory-shortage bottleneck, the proposed system can support real-time watermarking for multiple high-resolution (up to 1404 × 960) video.

1. Introduction

With the rapid growth of high speed access to the INTERNET and multimedia streaming technologies, copyright protection of digital media has become a major concern for content providers who support large scale media-on-demand systems. Copyright owners (e.g., movie studios) simply do not want to risk their material potentially being pirated after it is digitally transmitted, unless there is reliable content protection technology such as video watermarking [12–14].

Nevertheless, on-demand service for high-quality video has been greatly anticipated by both the research and industry communities. IEEE 802.16 [15], the recently proposed wireless broadband access standard (WiMax), has significant support for video-on-demand services. In WiMax (or other access networks), video is transported from a content storage facility to the head end, which further distributes the content to the subscriber. In this architecture, the video stream entering the head end is the original copy, and the video stream leaving the head end carries the embedded watermark signature for a particular subscriber. The head end node must first decode the incoming compressed video, embed the watermark signature into the video content, re-encode the video into the MPEG-2 stream format, and finally send it out to the WiMax downlink.

\* Corresponding author.
E-mail addresses: jwang3@vcu.edu (J. Wang), jcliu@cise.ufl.edu (J.C.L. Liu), masilelam@vcu.edu (M. Masilela).

0045-7906/$ - see front matter Published by Elsevier Ltd.
doi:10.1016/j.compeleceng.2008.06.011
In this paper, we present a real-time video watermarking system that utilizes a parallel MPEG-2 engine and a low-complexity, content-based watermarking algorithm. Our watermark embedding algorithm is intended for use at the local video distributor who directly delivers video to the end users (subscribers). In such systems, a common video content must be watermarked to distinguish between versions being delivered to many end users (subscribers). Such systems include traditional CATV, fiber-to-home, and the emerging WiMax systems. For traditional cable based video distribution systems, our scheme will be implemented at the local headend node (see Fig. 1). With our scheme, the headend node embeds subscriber-specific watermarks before the content is transmitted to the end user. This way, common digital content entering the headend will generate different video streams that contain unique watermark information. The watermark extraction and verification algorithm can be used to identify the source of illegal copies as legal evidence when needed.

We address two challenges for video watermarking in such applications: (1) the watermarking system must have robust detection response and must be resilient against collusion and temporal attack, and (2) the system must deliver real-time watermarking performance for broadcast level digital video (MPEG-2/4). To satisfy these two goals, our approach is based on embedding watermark signatures into the DCT block coefficients [13]. We use a drift compensation technique to reduce the watermark cross-talk between the predicted frames and the predicting intra-coded frame. To counteract temporal and collusion attacks, our method uses a novel block selection algorithm which dynamically selects DCT blocks for watermark embedding. The idea is to make the watermarked location randomly spread in the different video frames. With our method, the watermark location of video frames shows very small temporal correlation which make it safe against temporal and collusion attacks. Each frame is analyzed based on its local image content and its dependency on the I-frame.

To achieve real-time watermarking performance for compressed video, we designed a parallel MPEG-2 watermarking engine that can run on both cluster and multi-processor environments [11]. The parallel watermarking engine distributes the watermarking workload into several computing nodes using a Master/Slave architecture [11], where the master is in charge of data distribution/collection and the slave nodes perform MPEG-2 decompression, watermark embedding, and MPEG-2 encoding algorithms for the assigned task. By configuring a variable number of slave nodes, the system achieves scalable performance.

We observed an abnormal performance degradation (e.g., frame rate dropping from 20 to 2.5 fps) when watermarking high spatial resolution video. System trace data indicates that frame buffers eventually become a system bottleneck during high-speed operations. To address these challenges and obtain high scalability for high-quality video, we proposed and implemented two memory sharing mechanisms that reduce the buffer requirement. The first is Minimum Transmission Buffer in the slave nodes (ST scheme). In the ST scheme, we design a set of rotation rules to use a 3-frame transmission buffer to watermark one group-of-picture. To further reduce the buffer requirement in the slave node, we propose a dynamic buffer scheme which reduces 15% more of the effective frame buffers. With the revised buffer schemes, our system is able to deliver scalable watermarking performance based on the configuration of slave nodes.

The organization of the paper is as follows: Section 2 covers related studies. Section 3 describes the content-based watermarking algorithm. In Sections 4 and 5, we present the experiment results for watermarking high-level MPEG-2 video and two improved buffer management methods. Section 6 concludes this paper.

Fig. 1. Digital video distribution system with real-time watermarking embedding. Watermark is inserted at the head end node.
2. Related study

Many video watermarking methods have been proposed and evaluated in the past decade [12–14,20]. In JAWS (just another watermark scheme) [13], watermark information is inserted in the spatial domain. However, the watermark information is spread through several video frames, thus the detection response is weak if only one video frame is present. The JAWS scheme is also not very robust against low and moderate MPEG-2 compression. For compressed video, hiding watermark information in the DCT coefficients [14], Run-Length-codes and motion vectors have been discussed. However, detection for the last two methods is very difficult in the decompressed domain. Wu [12] and Podilchuk [16] proposed to use image local properties of HVS (human visual system) for hiding data in the block transformed image/video. Most of these works address the preservation of watermarks after the video is attacked by distortion/re-encoding and often assume that the watermark is embedded/extracted offline.

Hartung and Girod [14] discussed a compressed-domain-embedding scheme for the MPEG-2 video. Their method includes a drift compensation process to handle the watermark propagation in the predicted frame. However, verification of watermarks requires that the suspect video be perfectly synchronized to the original copy. A similar work is described by Alattar [20] to deal with watermark propagation in MPEG-4 video. Other interesting works include discrete wavelet transforms based watermarking [23]. Those schemes are particularly useful for video compressed at low-bit-rates and are suitable for wavelet coded image/video.

Two important merits in evaluating video watermarking algorithms are the detection robustness and watermark security against different attacks. Attacks based on spatial distortion (such as geometrical transformation) have been discussed by many authors [22,24,25]. Blind detection [27,29–31] is attractive in many scenarios since it does not require the original video content. However, it is also subject to cumulative and temporal attacks. In [21], collusion attacks are analyzed, where collections of video frames are combined to generate an unmarked copy of the original. For high-quality MPEG-2 video, we find collusion and temporal domain attacks constitute serious problem for the watermark security. Our proposed watermarking scheme will provide an effective mechanism to block both types of attacks.

Real-time watermarking for video broadcasting services requires high performance video codecs. Recent developments in high performance computing have made it possible for real time MPEG-2 decoding [2,4,8,10]. More dedicated multimedia instructions were introduced in many processor architectures, and were used to accelerate MPEG-2 decoding process [4,9,10]. Ravi [4] conducted a complete evaluation of MMX technology for Filtering, FFT, vector arithmetic and JPEG compression applications. With the Pentium III 700 MHz CPU, the main-level MPEG-2 video (DVD quality) can be decoded at nearly jitter-free quality.

However, these approaches could not support real-time processing of multiple MPEG-2 video streams which is very critical in some multimedia applications such as the video watermarking example discussed here. Moreover, these approaches strongly depend on specific hardware and are not flexible and reusable. In some case, a generic pure software solution is more favorable. As long been shown in literature, pure software MPEG-2 encoding/decoding requires large amounts of computation power. There are several works [1,3,6] on parallelizing the MPEG-2 encoding process, using SMP machine or workstation clusters. In [3], a data-parallel MPEG-2 encoder is implemented on an Intel Paragon platform and a real-time encoding performance is reported for low resolution video [5]. Hartung and Girod [14] discussed a compressed-domain-embedding scheme for the MPEG-2 video. Their method in-

3. Proposed watermark scheme

The basic structure of our proposed scheme uses a similar design [14] and is shown in Fig. 2: the MPEG-2 stream is partially parsed, DCT blocks are modified by a user specific watermark signature, and finally re-assembled back into a MPEG-2 transport stream before being sent to subscribers. Each VOD session requires one such processing block as shown on the left side of Fig. 2. Notice that the implementation detail of the processing block is not shown. The computing for each VOD session block inside the head end node is carried out by the parallel watermark engine consisting of the master node and a couple of slave nodes. In this section, we focus on the watermark embedding algorithm and its integration to the parallel MPEG-2 engine.

Our watermark algorithm uses a novel block selection algorithm to trade off the computation complexity, video fidelity, and watermark robustness. To minimize the impact on the image quality, the watermark for individual $8 \times 8$ DCT blocks uses an additive embedding method [16]. The computation overhead of the insertion procedure contains the following major steps: MPEG-2 header parsing, run-Length decoding for the DCT coefficients, de-quantization, block selection, adding the DCT transformed watermark signal, quantization, and run-length encoding. These steps are applicable for all frames. We will show how individual block are watermarked for all frame types.

In our application domain, watermark extraction is typically performed to identify the source of illegal copy. It is safe to assume that high-quality original video content is available. Due to this consideration, the main issue here is to be able to provide frame-to-frame linkage between the original video and the illegal copy. Such linkage will be very important when used as legal evidence.
3.1. Basic embedding: I-frame

3.1.1. Embedding process for DCT block

The embedding of the watermark for the I-frame is similar to the IA-DCT scheme as suggested in [16]. Using their terminologies (see Table 1), this procedure can be expressed as follows:

\[ X_{u,v,b}^* = \begin{cases} X_{u,v,b} + t_{u,v,b}^* w_{u,v,b} & \text{if } X_{u,v,b} > t_{u,v,b}^* \\ X_{u,v,b} & \text{otherwise} \end{cases} \]

The watermark sequence \( w_{u,v,b} \) follows the \( N(0,1) \) Gaussian distribution. \( t_{u,v,b}^* \), also known as just-noticeable-difference (JND), is the embedding depth for different frequency components calculated according to the human visual system [17].

Since the embedding depth is fully determined by the DCT coefficients of the \( 8 \times 8 \) block, the above embedding procedure can be performed in the compressed domain. The general structure of the embedding of the I-frame is shown in Fig. 2. For the sake of convenience, this embedding scheme is referred as IMODE in the rest of this paper.

3.1.2. Extraction process for DCT blocks

The corresponding detection procedure consists of: (1) extract the difference between original and watermarked frame in the (block) transformed DCT domain. \( w_{u,v,b} = X_{u,v,b} - X_{u,v,b}^* \), (2) the difference is then normalized \( w_{u,v,b} = w_{u,v,b} / t_{u,v,b}^* \). Since the original frame is available in the detection party, the normalization factor, which is the embedding depth \( t_{u,v,b}^* \), can be calculated, (3) after this, the similarity based detection statistic [18] \( p = w^* \cdot w / \sqrt{w^* \cdot w^*} \) is calculated.

In Fig. 3a, the correlation detection response for intra-coded frames is evaluated for a test MPEG-2 stream coded at 5 MBps. By varying the number of watermarked DCT blocks, we are able to describe the relationship between watermark length \( n \) and the detection performance. It is observed that the detection response increases quickly as the marked block number \( n \) increases. When \( n \geq 50 \), the correlation between the extracted watermark and the original stabilizes at 0.75 or higher, which strongly supports the presence of the expected watermark. Our experiments also confirmed that the correlation response is lower than 0.1 if the tested video frame is not marked by the testing watermark sequence.

Fig. 3b shows the degradation of image quality as the number of embedded DCT block increases. When less than 50 blocks are marked, there is virtually no quality loss. However, the PSNR plot in this area shows a counter-intuitive increase as more blocks are marked. The exact reason of this behavior is not clear at this moment, but it surely provides more evidence that the PSNR alone might not reflect the true image quality. When more than 50 blocks are marked, we observe a steady decrease in the image PSNR. With 100 blocks being modified, a 6 db loss in PSNR is observed.

---

**Table 1**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u, v )</td>
<td>Matrix coefficient index (of a ( 8 \times 8 ) DCT block)</td>
</tr>
<tr>
<td>( b )</td>
<td>DCT block number within an image</td>
</tr>
<tr>
<td>( X_{u,v,b} )</td>
<td>The ( (u,v) )th DCT coefficient at block ( b )</td>
</tr>
<tr>
<td>( W_{u,v,b} )</td>
<td>Watermark bit to be embedded for ( X_{u,v,b} )</td>
</tr>
<tr>
<td>( X_{u,v,b}^* )</td>
<td>The watermarked ( (u,v) )th DCT coefficient at block ( b )</td>
</tr>
<tr>
<td>( t_{u,v,b}^* )</td>
<td>Just-noticeable-difference (JND) threshold for ( X_{u,v,b} )</td>
</tr>
</tbody>
</table>
3.2. Embedding procedure for predicted blocks in predicted frames

For the sake of convenience in this discussion, we define some useful notations in Table 2. The spatial relationship between the predicted DCT blocks $I_1$ and its reference block $I_0$‘s are illustrated by Fig. 4. Without loss of generality, we focus on forward coded blocks here. The discussion in this subsection can be easily extended to the bi-directional predicted blocks in the B-frame with minor modifications.

Let $U_i; (k; h)$ be the watermarked version of $I_i; (k; h)$. Based on the discussion in the previous subsection

$$ U_1; (k; h)(z) = \begin{cases} I_1; (k; h)(z) + J_1; (k; h)(z)w_k; h(z) & \text{if } I_1; (k; h)(z) > J_k; h(z) \\ I_1; (k; h)(z) & \text{otherwise} \end{cases} $$

(2)

to compute $I_1; (k; h)$, we need the residual block $R_1$ and the prediction block $I_0$ from the predicting frame. Specifically, we have

$$ I_1; (k; h) = I_0; (x; y) + R_1; (k; h) $$

(3)

$R_1$ is readily obtained after run-length decoding for the current frame, however, $I_0; (x; y)$ is not. It is frequently observed that $(x; y)$, the location of the best-matched-block in the predicting frame for $I_1; (k; h)$, is not aligned to the 8 by 8 boundary as shown in Fig. 4. Since all coded blocks are aligned to the 8 by 8 grid, the non-aligned predicting block $I_0; (x; y)$ cannot be extracted from the compressed bitstream directly. To calculate $I_0; (x; y)$, the four surrounding blocks $I_a; 0, I_b; 0, I_c; 0, I_d; 0$ should be decoded to provide the surrounding area containing $I_0; (x; y)$, then a 8 by 8 luminance block can be extracted to calculate the forward DCT $I_0; (x; y)$.

Having $I_1; (k; h)$ available from Eq. (3), the local JND matrix $J_1; (k; h)$ can also be calculated according to [17]. The modulated watermark is then $W_1; (k; h) = J_1; (k; h)(z)w_k; h(z)$.

It should be pointed out that the actual predicting block, denoted by $U_0; (x; y)$, used at the decoder is the ‘watermarked’ version of $I_0; (x; y)$. Here $U_0; (x; y)$ can be calculated similarly to $I_0; (x; y)$, except that all operations are based on the watermarked predicting frame.
Ib
Ia

Techniques which undo spatial distortion (such as geometrical transformation) [24,25].

be resilient to both spatial and temporal attacks. For spatial attacks, a commonly used counteract is synchronization tech-

original video content at the watermark detection/extraction. A common requirement is that watermarking schemes should

3.3. Watermark detection considerations

Table 2

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_{1,(k,h)})</td>
<td>An 8 by 8 block of the predicted frame, with left-up corner at ((k,h)). (k) and (h) are multipliers of 8</td>
</tr>
<tr>
<td>(R_{1,(k,h)})</td>
<td>The 8 by 8 residual DCT block in the predicted frame, corresponding to (I_{1,(k,h)})</td>
</tr>
<tr>
<td>(I_{0,(x,y)})</td>
<td>The 8 by 8 predicting block for (I_{1,(k,h)}) in the reference frame, with left-up corner at ((x,y))</td>
</tr>
<tr>
<td>(x)</td>
<td>The block contain left-up corner of (I_{0,(x,y)}) in the reference frame</td>
</tr>
<tr>
<td>(y)</td>
<td>The block contain right-up corner of (I_{0,(x,y)}) in the reference frame</td>
</tr>
<tr>
<td>(k)</td>
<td>The block contain left-bottom corner of (I_{0,(x,y)}) in the reference frame</td>
</tr>
<tr>
<td>(h)</td>
<td>The block contain right-bottom corner of (I_{0,(x,y)}) in the reference frame</td>
</tr>
</tbody>
</table>

![Image](image.png)

**Fig. 4.** Motion prediction and compensation.

Finally, we have the modulation algorithm for the residual block in the P-frame

\[
\hat{R}_{1,(k,h)} = \phi_{1,(k,h)} - \phi_{0,(x,y)} = I_{0,(x,y)} + R_{1,(k,h)} + W_{k,h} - \phi_{0,(x,y)} = R_{1,(k,h)} + W_{k,h} + (I_{0,(x,y)} - \phi_{0,(x,y)}).
\]  

(4)

3.3. Watermark detection considerations

Watermark schemes are largely divided into blind-detection or non-blind-detection schemes based on the availability of original video content at the watermark detection/extraction. A common requirement is that watermarking schemes should be resilient to both spatial and temporal attacks. For spatial attacks, a commonly used counteract is synchronization techniques which undo spatial distortion (such as geometrical transformation) [24,25].

Blind watermark detection techniques assume that the original image/video is not available at the detection time. Most such works rely on certain spatial invariance and correlation-based detection. Blind image watermarking methods use Fourier–Melin transform [26], geometric invariants cross-ratios [27], self-reference [29], and spread spectrum embedding [28].

A drawback of blind watermark methods is that they typically embed strong and excessive amounts of watermark bits, in compensation for the lack of original image, to assure robust detection. This however leaves the watermark more vulnerable to cumulative attacks, since the same watermark information is plentifully presented in the video.

Furthermore, video watermarking schemes must address temporal attacks such as frame re-ordering and frame dropping. Establishing synchronization would require tremendous computing time to detect the embedded watermark, particularly when a blind watermark method is used. Some proposed methods use frame dependent watermarks [30,31] to take advantage of the redundancy to reduce the search required for synchronization. Others [19,22] use a queue and a state predictor to perform a search to establish and maintain temporal synchronization.

We use a different approach that is based on non-blind detection: (1) we assume that original video is available at the detector to allow a less invasive embedding strength, (2) the embedding location for each frame is highly correlated on the image content of the frame and control parameters, (3) the embedding location can be computed precisely and efficiently by the detector from the original video, (4) the embedding location can be estimated from the watermarked video frames to assist time synchronization at the detector. By “randomizing” the watermark location using image content, it is virtually impossible for the attackers to remove the watermark without the original video and control parameters used during watermark embedding. A high-level abstraction of the detector is shown here:

3.3.1. Watermark detection (F)

(1) Compute the estimated watermark location vector \(L\) using the image content of the watermarked video frame \(F\) under test.
(2) Match \(L\) against the watermark location for the original video and obtain an estimated frame index \(i_F\). The frames with the most overlapping in the watermark location are potential synchronization points.
(3) Use \(i_F\) as the reference frame and compute the watermark signature \(W_{i_F}\).
(4) Apply correlation detection on \(W_{i_F}\) and report the matched watermark signature.
3.4. Content-based block selection

We now focus on the determination of watermark locations based on video content. To combat the cumulative attacks, a safe watermarking scheme must distribute watermark signatures into different blocks along the video sequence. For example, the embedder software/hardware might randomly decide watermark locations and store these locations locally. At the extractor, this information will be retrieved and used during verification. However, this requires frequent downloading of watermark locations and strict anti-leaking measurements. Thus a better scheme is to select image blocks based on the image properties and hard-code the block selection algorithm in the extractor.

Several researchers [16] have suggested to select image blocks with rich contents. This was originally proposed to minimize the video quality degradation due to watermarking. We believe the content-based concept might provide a dynamic block pattern which is desired to defeat cumulative attacks. A good indication of image complexity is image variance. A large variance usually indicates a complex pixel pattern, large AC coefficients in mid-ranges, and thus can bear more watermark noise. However, if only local variance is used for block selection, the resultant watermark blocks will have significant overlapping between video frames within the same scene. Further more, if a video frame contains a complex background color, these background blocks will dominate the selected image blocks. These background image block are often well predictable from the I-frame and the resultant residual blocks have small intensity and variance, thus are more vulnerable under the cumulative attack.

The method we suggest is to jointly consider the motion information and the residual block variance in the block selection algorithm. The principle is to select image blocks that mostly distinguish current video frames from their predicting frame. Thus high priority is given to image blocks that can not be predicted from previous video frame. If the video is relatively static and there are insufficient amounts of intra-coded blocks with high variance, priority should be given to blocks of the moving objects. Such image blocks often have high motion vectors. The selection algorithm is described by the following pseudo-code.

3.4.1. Motion-variance block selection

(1) Calculate the residual image by subtracting the candidate video frame from the predicting frame using motion parameters.
(2) For each block $i$ in the residual image:
   (a) calculate the variance value $v_i$,
   (b) calculate the moving index $u_i = 1/8\sum_{k=\text{neighbor}} |m_k|$, here $m_k$ is the motion vector of the block $k$,
   (c) calculate the weighted quantity $U_i = w \cdot v_i + (1 - w) \cdot u_i$ where $0 < w < 1$ is a weight parameter. A video clip with slow motion should adopt a high $w$ value.
(3) Sort the image blocks in descending order of $U_i$. The first $N$ blocks will be selected for watermarking.

For each video frame, the above algorithm needs to compute the residual block variance and an average motion vector for all DCT blocks. Both block variance and motion vectors are by-products of the MPEG-2 encoding process, thus can be directly used for block selection. If the video is already compressed, the motion vectors are available. The main computation overhead will be in the computation of the block variances. Fig. 5 shows the image blocks selected according to the above algorithm.

3.5. Integration with parallel MPEG-2 engine

The watermark scheme seamlessly integrates into a parallel MPEG-2 engine to watermark multiple video streams in real time. The parallel watermark engine consists of one master node and many slave nodes. The number of slave nodes is configurable based on the required performance.

For each incoming video stream, the master node performs partial MPEG-2 decoding and assigns a group of pictures to the slave nodes for watermark embedding. Each slave node performs the block selection algorithm and modifies the selected DCT blocks using the user-specific watermark signature. The results are sent back to the master node, where the MPEG-2 transportation streams are regenerated. The parallel engine requires an UNIX-type OS and message passing interface (MPICH). The system has been tested on both cluster environment and multiple-processor machines.

The high-level watermarking process and the interaction between the master–slave nodes is described here:

- **Master node**: The back-end MPEG-2 parser will parse the MPEG-2 transportation header and syntax, and frame DCT blocks are extracted.
- **Master node**: The middle-end watermark embedder will assign a group of partially parsed pictures to an idle slave node. The DCT blocks and the watermark signature sequences for $N$ subscribers are sent to the slave node.
- **Slave node**: For each of the unmarked pictures, the block selection algorithm is performed. These blocks are modified using the corresponding watermark bits for each subscriber. In this step, each selected block generate $N$ variations for the $N$ subscribers. The marked DCT blocks are sent back to the master node.
Fig. 5. Variance based block selection: (1) original video frame, (2) select the top 1000 blocks, (3) the top 100 blocks for the first P-frame, and (4) the top 100 blocks for the second P-frame.
• **Master node:** The last stage executed at the master node is the front-end MPEG-2 stream re-generating. In this stage, the unmarked DCT blocks and the marked DCT blocks from the slave nodes are re-coded into MPEG-2 transportation stream, which are fed to the transmission cable to the subscriber end.

To optimize the system performance, the watermark algorithm and the decoding and encoding algorithms share as many computing subtasks as possible. In step (1), the MPEG-2 parser is only executed once per frame. In step (3), the result of the block selection algorithm is also reused in the slave node to eliminate repetitive computations. Notice that the major computing task is on the slave side where an unmarked picture must be watermarked \( N \) times. As can be see from the above, the system involves large amount of data exchange, which could have significant impact on the overall realtime performance. We will focus on the optimization of frame buffers in the next two sections.

4. **Performance evaluation**

This section will first demonstrate the detection performance of our watermark method against temporal attacks. We will then show the realtime watermarking performance of the system under different loads. The test video is encoded with \( N = 12 \), and \( M = 3 \) with chroma format 4:2:0. Each video content has seven versions with different resolutions, from \( 352 \times 240 \) to \( 1404 \times 960 \). The intermediate resolution is chosen such that a continuous performance trend can be observed. We use the same GOP structure, quantization table, color format, and motion search range as the encoding parameters to have a fair comparison. Each encoded video consists of 60 frames, which is roughly four GOPs.

4.1. **Watermark robustness evaluation**

In Fig. 6, the watermark detection performance of an off-synchronized video segment and the result after our re-sync process are shown. In this experiment, the same watermark is embedded into all video frames. We use a video clip with slow motion. For each video frame, marked DCT blocks are individually selected by the Motion-variance block selection (MVBS) block selection algorithm. The resultant watermark locations shows little overlap among consecutive frames. Before watermark extraction and detection, we introduced an artificial frame offset to simulate frame dropping temporal attack. An offset \( d \) means the detector will use frame \( i + d \) as the original for suspect frame \( i \).

**Fig. 6a** shows the average detection response with a one-frame sync error. An interesting observation is that the detection response decreases as the number of watermark blocks increase. This behavior is somewhat surprising, since the previous experiments (with no temporal attack) all show that detection response becomes stronger as watermark blocks increase. Further examination of the watermark location shows that there is a strong correlation among the first (strongest) watermark locations. These locations are strong enough such that they are relatively static even with a small frame offset \( (d = 1) \). However, when large numbers of watermark blocks are used, the correlation between the time-shifted watermarked video and the original video becomes significantly weaker and fails detection.

**Fig. 6b** shows the detection response after re-synchronization for the attacked video with different frame shifts. The four cases corresponding to the four curves represent the detection response for \( d = 1 \) and \( 2 \) before and after re-synchronization. The watermark block number is from 100 to 1000. We clearly observe that the detection response quickly decreases as the frame sync error increases. With a video offset by more than two frames, the detection response dropped from strong (higher than 0.6) to very uncertain (low than 0.3). The top two curves, corresponding to the re-synchronized detection response, are almost as good as perfect synchronization. In summary, temporal attacks result in wide variation of watermark locations, and the detection response shows a very low correlation to the expected watermark sequence. Video re-synchronization based on this technique turns out to be very effective against temporal attacks.

4.2. **Experiment setting and results**

Using the performance model in [11], the expected watermarking frame rate can be approximated by

\[
FRD = \frac{(N + D)}{\max(D \cdot T_{\text{single}} + T_{\text{ms}} + T_{\text{sm}} + 2c, N \cdot (T_{\text{sm}} + T_{\text{ms}} + 2c))}
\]

here \( N, \) \( D \) denotes the engaged processor number and the length of GOP. \( T_{\text{single}} \) is the time required to watermark one frame. \( T_{\text{sm}} \) and \( T_{\text{ms}} \) are the transmission time between the master and slave nodes. Using the same hardware configuration as in the SUN SMP environment in [11], the expected watermarking performance is showed in the Table 3.

Though the expected performance can be predicted via our proposed performance model, it is not clear whether the experimental results will totally agree with our prediction. By using a SUN SMP machine with fourteen 248-MHz UltraSparc CPUs, 512-MB memory space, we have collected the scalability performance with different video resolutions. With a faster CPU running at 3 GHz and more memory up to 2 Gigabytes, the head end node will be able to support five to 10 concurrent VOD streams. The testbed in our experiments is dedicated to one VOD stream.

The achieved frame rates for the low- and main- level MPEG-2 video are very close to the prediction. The results showed only a slight difference among the three video contents.
For the small resolution video (Fig. 7a), we observed a linear increase of frame rate. The maximum frame rate is achieved when 14 nodes were deployed, providing an average of 220 f/s. Since the video size is small, the system’s theoretical peak could reach 500 f/s (at 30 nodes) according to the prediction in [11]. Our test platform only has 14 nodes, thus the saturation point will not be reached. The results for the main-level video (720/C2 480) also conforms with the prediction. The performance for the three video titles shows little difference in terms of the frame rate. Each of them increases in a close to linearly fashion when more slave nodes are used. The highest frame rate achieved is 70 f/s (with 14 nodes).

However, the scalability performance for the high-resolution MPEG-2 videos are not satisfactory. In Fig. 8b, the frame rates for (1408 × 960) MPEG-2 files are illustrated. Starting with 2 f/s at a single node configuration, a linear increase can be observed. The highest performance is about 20 f/s. At 10 slave nodes, the frame rate for “flower” dropped to 2.5 f/s. For “tennis” and “calendar” test video, a similar performance degradation is observed at 12 slave nodes.

4.3. Memory usage analysis

The performance degradation shows our pipeline scheme is bounded by another system bottleneck. Analysis of machine statistics narrows down the problem to memory shortage due to heavy loads. We observed that the data exchange between

<table>
<thead>
<tr>
<th>Video spatial resolution</th>
<th>2 Nodes (f/s)</th>
<th>4 Nodes (f/s)</th>
<th>8 Nodes (f/s)</th>
<th>16 Nodes (f/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>352 × 240</td>
<td>50</td>
<td>600</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>704 × 480</td>
<td>18</td>
<td>40</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1024 × 1024</td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>1404 × 960</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>25</td>
</tr>
</tbody>
</table>
the master node and slave node is based on GOP, which usually contain 10 to 20 frames. The slave node has to allocate buffer space to accommodate both the compressed and decompressed video frames, for each of the GOP. In the master node, a dedicated buffer space (1 GOP) is reserved for display/outgoing data and another for receiving data from the slave node. The following Fig. 9 depicts the buffering requirements and relations between the master and slave nodes:

As shown in Fig. 9, the amount of memory required at the master node $M_m$ consists of several parts:

- $m_c$ for the executable code, which is about 500 kB,
- $m_{\text{streambuffer}}$ used as the streaming buffer to receive the compressed video streams from the video server, fixed at 1 MB,
- $m_{\text{outbuffer}}$ and $m_{\text{inbuffer}}$ for exchanging frame data between the master node and slave nodes during parallel decoding and watermarking.

Notice that $m_{\text{outbuffer}}$ must be large enough to accommodate one GOP of MPEG-2 compressed frames. Similarly $m_{\text{inbuffer}}$ needs to accommodate two GOP of watermarked raw frames (one GOP for displaying and another for incoming traffic). The memory requirement for the master node is thus:

$$M_m = m_c + m_{\text{streambuffer}} + m_{\text{outbuffer}} + m_{\text{inbuffer}}.$$ 

The amount of total memory also depends on the resolution of video being processed. Using the horizontal video size $h$ and vertical video size $v$, GOP = 15, and the average compression ratio $\lambda$ is 20, we have:

$$M_m = 0.5 + 1 + \lambda * h * v / 10^6 + 2 * \text{GOP} * h * v / 10^6 \text{ MB} = 1.5 + (\lambda + 2 * \text{GOP}) * h * v / 10^6 \text{ MB}.$$ 

For the slave processes, the size of executable code is also 500 kB. A compressed data buffer is used to receive data from the master (the same size as the $m_{\text{outbuffer}}$ in master node). The transmission buffer can serve two purposes, it is used during the decoding process, and thus needs separate space for the YUV components of each macroblock, then an in-place transmission.
can be done without moving data. Using a 4:2:0 color scheme, the average bits per pixel is 12 bits, instead of 8 bits used in the display system, thus the transmission buffer $m_t$ is 1.5 times the size of $m_{inbuffer}$. We have

$$M_s = m_c + m_{compressed~buffer} + m_{transmission ~buffer} = M_c + m_{outbuffer} + 1.5 \cdot m_{inbuffer}.$$ 

Using the above two equations, we can calculate the memory requirement for the master node and the slave node for each video format. For 1404 $\times$ 960 testing video, $M_m = 42$ MB, and each slave node needs about 30.8 MB. The original consideration of a pipeline parallel design is to minimize the communication cost and reduce the number of high-level network access times. However, the memory buffering scheme is not considered to be optimized.

The cumulative buffering space will grow quickly when using a large scale slave-node configuration, which cause unsatisfactory scalability performance when the number of slave nodes is large. For instance, let $N$ be the number of slave nodes, the total memory requirement becomes

$$M_t = M_m + N \cdot M_s.$$ 

Using the parameters of our MPEG-2 video, the actual memory used is listed in Table 4. In Fig. 8a and b, the frame-rate drops at $N = 9$ and 11 for two testing video streams. The corresponding amount of memory used is 319.9 and 296.5 MB, respectively. The minimum of these two should be used as the indication of potential memory shortage. This amount of memory is actually 70% of the system physical memory.

### 4.4. Impact of memory shortage

A non-optimized buffering scheme will have an effect on the competition between user processes (e.g., our communication and decompression software) and system processes (e.g., demand-paging mechanisms by OS). Because of the shortage of memory, the system processes will generate a significant number of page faults, which in general slows down the decompression speed. The shortage of system memory will force the operating system to swap some of the memory pages out to hard disk, and this activity in turn will use more CPU time, thus affecting the performance of all user space processes.
The CPU time distribution and the number of page faults is collected to verify our theory. Fig. 10 illustrated our measured number of page faults vs the number of slave nodes. For the sake of clarity, we only present the results for “tennis”, the “flower” and “calendar” showing similar results for this measurements.

The following observations can be made:

- For the 352 \( \times \) 240 video, the page faults virtually remain unchanged, and were kept in a low level (1010 page faults/frame). Increasing the video resolution to 704 \( \times \) 480 is effectively reflected by the rise in the number of page fault, a four fold jump is observed. Nevertheless, the 704 \( \times \) 480 case still has a flat curve even with the increasing number of slave nodes, indicating the system is running steadily.

- For the 1024 \( \times \) 1024 video, the number of page faults increase considerably at beginning, but still within a manageable level. About 1200 page faults per frame are observed for 2 slave nodes, and remain the same until 9 slave nodes. This is followed by a drastic increase at 10–12 slave nodes, reaching 3500 page faults per frame at 12 slave nodes as the peak. Then the figure drops back to a certain degree, but is still maintained at a high level (more than 2500). Compared to the results in Fig. 8b, high-page-faults overlap with the low-frame-rate points. The page-faults behavior of 1404 \( \times \) 960 video shows the same pattern as in the 1024 \( \times \) 1024 case. The highest page-faults reached 4700 faults/frame at 10 slave nodes, where the frame rate drops to 2.5 fps (Fig. 8a).

![Fig. 9. Memory usage illustration.](image)

**Table 4**

<table>
<thead>
<tr>
<th></th>
<th>Memory requirements for different nodes (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Horizontal resolution</td>
</tr>
<tr>
<td>Tennis40</td>
<td>1404</td>
</tr>
<tr>
<td>Tennis60</td>
<td>1024</td>
</tr>
</tbody>
</table>
Fig. 10b shows the overall CPU usage distribution between user space process (watermarking engine), system cost (paging), and system idle time. With one slave node, 90% of the system time is idle, 8% of the CPU time is used in the user space, and the rest, 2%, for other system maintenance. With the increasing number of slave nodes, the user space time increases proportionally, and the system idle time decrease. During this period, more CPU time is used for the slave nodes, and the frame rate increase linearly. After 8 slave nodes, however, both system idle time and user space time dropped significantly, while the system overhead showed a significant increase. Ninety percent of the CPU time is used by the operating system, while user space only occupies 5% of CPU time. We conclude that the system spends most of its CPU time swapping page in/out instead of watermarking frames.

5. Performance improvement with buffering sharing

5.1. Slave-node buffer reduction

To reduce the memory requirement in slave nodes, the buffering scheme should be redesigned. The task is complicated by the decoding dependency inside the MPEG-2 video structure. To decode a B-frame, we need at least two reference frames. This indicates that the worst case of the minimum buffer should be three frames, with two frames for reference frames, and one for the working B-frame. With a careful redesign of the master–slave communication protocol, using a 3-frame transmission buffer in the slave side is possible, and we call it the ST scheme. When the picture size is $1024 \times 1024$ and GOP = 15, we can save about 12 MB buffer space per slave node, about 80% reduction in the slave side.
To adopt the proposed memory-efficient algorithm, the slave watermarking process needs to rotate the usage of two reference buffers. Let forward and backward point to the forward predicting frame, and backward point to the backward predicting frame from the view of B-frame decoding, the following rotation rules must be obeyed:

- The first frame, I-frame, must be decompressed to the forward, which initially points to the first buffer.
- The first P-frame will refer to forward, and must be decompressed to backward, initially pointing to the second buffer.
- The following P-frames will use backward as reference frame, and are decoded into the forward. After completion, forward and backward have to be switched.
- Each B-frame will refer to forward as forward predicting and backward as backward predicting frame. The decompressed data is stored in the third buffer.

On the master side, the master should be able to receive the data whenever it appears in the network layer, such that the slave nodes don’t have to wait on the blocking transmission. This can be implemented via the use of UNIX SIGNAL mechanism.

The minimum number of required frames-per-buffer can be further decreased from 3-frames to 2 frames. For the I- or P-frames, we need one buffer for prediction picture, and another buffer for the working frame. The two buffers change their role after watermarking, one I- or P-frame, such that the most recently processed I- or P-frame is the next prediction frame. For the B-frame, we can directly send the watermarked block through the MPI protocol without buffering it locally. The above discussion assumes the reference frame of P-frame is always the last decoded P-frame, and the reference frame for B-frame are the last two P-frames.

With this scheme, the new memory requirement becomes

\[
M' = N \cdot M_c = N \cdot (m_c + 1.5 \cdot 3 \cdot m_{frame} + m_{inbuffer}).
\]

Here the buffer space of the master node remains the same. For different types of video content, the expected amount of \(M'\) is plotted in Fig. 11.

It can be observed that the memory required increase at a slow slope, where each slave node will introduce only about 6 MB of additional space for the tennis40 video.

5.2. Implementation and experiment result

We implemented the ST memory allocation scheme, and repeated the experiments for the high-level MPEG-2 video with the new buffer management policy. The measured actual memory requirement is depicted in Table 5. The percentage of the new memory requirement and the original memory size is given below the actual memory size. The memory requirement for the ST scheme is significantly less than the original scheme, especially when more slave nodes are deployed. For one slave node, we need 53.5 MB for 1404 × 960 video, which is 27% less than the original one. For the 4-slave-node case, the ST scheme use 85 MB instead of the original 165 MB, which is almost a 50% memory savings. This closely matches the analytical estimation in the previous section, since the ST scheme can save 66% memory in slave node. The overall saving will always be at least 66% in total. The similar memory requirement is found for the 1024 × 1024 case. The ST scheme requires a maximum of 200 MB when all 14 nodes are utilized. Table 5 shows the memory usage when ST scheme is used.

Fig. 11b shows the average page fault caused by slave node during the decompression of 60 frames per node. The same video files used in Section 3 are tested. We observed that:

- The number of page faults for each individual node is significantly reduced, comparing to the number in Fig. 10. For the 1404 × 960 case, the number of page faults shrinks from 1500 to 1200, at one slave-node configuration. For 1024 × 1024 video, the page faults is now 943, 25% less than before.
- For all of the video streams, the number of page fault almost remains unchanged when increasing the number of the slave nodes. This phenomena is also observed in Fig. 8b before the memory saturation point. The flat curves shows that the system memory usage is still under “control”. The page-fault problem for the two high resolution video streams are eliminated, showing that the ST scheme has successfully relieved the memory bottleneck.

For each of the video sizes, we compare the performance for the three video titles mentioned early. Fig. 12b shows the scalable watermarking frame rate for 1404 × 960 video with ST scheme. We observed a close to linear increase in frame rates. For one slave node, we have 1.7 f/s for tennis and 1.86 f/s for flower. As the slave nodes increase, the achieved frame rate increases proportionally with slight differences between the three test videos. The peak watermarking rates are obtained at 14-slave nodes, where 20 f/s is observed.

For the two high-resolution video formats, our ST scheme successfully solved the memory-shortage problem. The observed near real-time frame rate shows that our scheme works well for high-quality video up to MP@HL video.

5.3. Further optimization in slave node

In the above section, we use a 3-frame transmission buffer for each slave node, which has already obtained significant reduction in the memory requirements. We also showed that the memory shortage will not happen until there are more than
32 slave node participating the parallel decompression, for the 1404 × 960 case. The maximum allowed number of slave nodes for other video size can be derived similarly based on the memory budget and the video resolution. This result can be easily extended to the multi-layered MPEG-2 video, where enhanced video stream layers may exist. These additional layers might include SNR scalability, DATA partition scalability and other scalability features. For these scalable MPEG-2 streams, we usually have $L$-layer sub-stream. Each requires the same amount of buffer space as that of a base layer in order to be successfully decoded. Assume a 3 layer MPEG-2 video is to be watermarked; the decoding/transmission buffer in the slave node will nearly be tripled. It can be expected that the memory shortage will become a bottleneck again. To further reduce the buffer requirement in the slave node, we propose a dynamic buffer requirement scheme.

It is observed that the watermarking procedure in the slave node does not need three frames all the time. More specifically, the I-frame did not refer to any other frames, thus we can only use one frame in the decoding/transmission buffer. Similarly, P-frame only refer to one frame (I- or P-frame), thus the total need for watermarking a P-frame is two. Only
B-frame needs all the three frame buffers. The total amount of buffers needed can vary during the watermarking process at
the slave node. Since all the slave nodes share the physical memory and perform decompression independently, we can
effectively reduce the memory requirement by dynamically allocating buffer in the slave node.

Let the ratio of I-, P-, B-frames in a GOP structure be $a:b:c$, the effective buffer space for one layer is expressed by

$$M = \frac{1}{3} \left( \frac{1}{a} + \frac{2}{b} + \frac{3}{c} \right) \left( a + b + c \right).$$

(5)

In a typical GOP structure of “IBBPBBPBBBPBBPBB”, we have $a:b:c = 1:4:10$. This will result in an effective buffer number of $39 / 15 = 2.6$, which is about 85% of the 3 frame buffer scheme. The effective buffer space is a function of the GOP structure. When the
percentage of the I-frame increases, the effective buffer space will decrease. In the extreme case of all I-frame GOP, the effective
buffer is 1 frame/GOP. A long GOP structure with many B-frame will make the effective buffer space approaching the limit, which is 3 frame/GOP. Assume a two-layered scalable MPEG-2 stream with 1404 × 960 video size is to be decoded, the memory require-
ment of the slave node is $M_s = \frac{2}{3} \times 1404 \times 960 = 18.7$ MB. Still assuming a 300 MB total memory budget and $M_m = 42$ MB, the
system can support up to $\frac{300 - 42}{18.7} = 14$ slave nodes. Further assume the watermarking time for such a two-layered high-resolution stream is twice as high compared to the one-layer stream in a purely serial software decoder, the 14-slave-node configuration can only produce up to $14 \times 0.9 = 12.6$ f/s. In order to have a higher performance, we need to further decrease the
buffer space, such that more slave nodes can be supported. 

Algorithm 1 (Dynamic buffer allocation in slave node).

```c
/* Three Buffers outbuffer[1,2,3] are used repeatedly. */
/* forwardb, reverseb, and currentb point to the forwarding reference */
/* frame, backward reference frame, and working frame, respectively */
/* The following steps decode and transmit one GOP frames */
RecieveGOP(&compressedBuffer)
allocate outbuffer[1,2,3]
for each frame f(i) in compressedBuffer do
```

Fig. 12. Watermark frame fate with buffer sharing: (a) 1024 × 1024 and (b) 1404 × 960.
if $f(i)$ is I-frame then
deallocate outbuffer[2,3]
currentb=outbuffer[1]
perform MPEG-2 I-frame decompression
transmit the outbuffer[1] to master
else if $f(i)$ is P-frame then
forwardb:=currentb, currentb:=outbuffer[2]
perform P-frame decompression
transmit the outbuffer[2] to master
else if $f(i)$ is B-frame then
reverseb=outbuffer[2]; currentb=outbuffer[3];
perform B-frame decompression
transmit outbuffer[3] and release it.
end if
end for
de-allocate outbuffer[1,2,3]

In fact, the concept of dynamic buffer allocation can be applied inside the watermarking of each frame. Since the decompression of each frame is based on a series of macroblocks, the overall buffer space can be reduced by dynamically allocating buffer space for the macro-blocks. For example, when watermarking the first macro-block, we only need to allocate a 16 x 16 block space. The other macroblocks buffer will be assigned when it is needed for watermarking. The buffer will grow as the slave-node processes more macroblocks, and it will reach the full buffer size after the watermarking is finished. After the watermarked frame is sent back to the master node, the decoding buffer can be released and the process repeats for the next frame. With this dynamic memory allocation scheme, we expect an additional buffer reduction of 0.5 frames. Notice that this scheme cannot reduce the amount of buffer space for the reference frame. The effective buffer requirement becomes

$$M = ((1 - 0.5) \times a + (2 - 0.5) \times b + (3 - 0.5) \times c)/(a + b + c).$$

Using the same GOP structure as above, the effective buffer size in slave node is 2.1 frames, which is about 60% of the 3-frame buffer scheme. The performance difference of the watermarking process with/without dynamic memory allocation is depicted in Table 6.

### 6. Conclusion

We described a real-time video watermarking scheme based on a parallel MPEG-2 engine. Our scheme embeds subscriber-specific watermark signature to the 8 x 8 DCT blocks of the compressed video stream. We use a content-based block selection algorithm to counteract the cumulative and temporal attack by randomizing and spreading the marked block location. Our scheme accomplishes a good tradeoff between robustness and efficiency for moderate and high-quality MPEG-2 video.

Our watermark algorithm is seamlessly integrated into a high performance MPEG-2 engine for realtime performance to satisfy the single input-stream, multiple watermarked output streams requirement in digital video delivering systems. Two buffer sharing mechanisms are studied to support multiple high-level high-profile MPEG-2 video streams. We propose the ST buffering scheme with a dynamic allocation algorithm to significantly reduce the memory demand within this parallel watermarking software. The performance results are very promising. Our experimental results show that the proposed parallel watermark system is robust against various temporal attacks, and can provide satisfactory performance for large scale video distribution networks.

### References


Ju Wang is an assistant professor in the department of Computer Science at Virginia Commonwealth University. His research focus is computer networks, wireless communications and multimedia systems. He has investigated issues in WCDMA, next generation cellular networks, and wireless sensor networks.

At 2003, He earned a Ph.D. degree from Computer, Information Science and Engineering department of the University of Florida. He is an active member of the IEEE Computer Society. He has reviewed papers for IEEE Transactions for Communication, Wireless and Mobile Computing, and IEEE Transaction of Wireless Communications.

Jonathan C.L. Liu received his Ph.D. degree in Computer Science from University of Minnesota. In 1993–1996, he had associated with the Distributed Multimedia Research Center (DMRC) at University of Minnesota, where he had R&D cooperations with Honeywell Technology Center (HTC) and IVI Publishing. In 1996–1999, he was an Assistant Professor of Computer Science with the School of EECS at Washington State University. Since August 1999, he has been with the CISE Department at University of Florida, where he is currently a tenured Associate Professor. His current research interests include high-speed wired and wireless networks, multimedia communications, parallel processing and artificial intelligence. He is a recipient of the National Science Foundation CAREER Award. He is a senior member of IEEE and a professional member of ACM.
Mbonisi Masilela graduated from Virginia Commonwealth University in December 2007 with a M.Sc. in Computer Science. Whilst at Virginia Commonwealth University, he was actively involved in a number of research groups including the NASA Research Partnership/MITAC where his efforts were concentrated in the areas of Embedded Systems Design, Wireless Communication and Biomedical Sensor Integration. In addition, he was also involved in Wireless Sensor research and design. He received his Bachelor of Science in Computer Science from Indiana University of Pennsylvania. His research interests include wireless communication and sensor networks, embedded systems design and data communication protocol design.