Journal of Visual Communication and Image Representation **12**, 1–16 (2001)
doi:10.1006/jvci.2000.0459, available online at http://www.idealibrary.com on IDEAL®

On-line Scene Change Detection of Multicast Video

Wensheng Zhou[1]

Information Sciences Lab, HRL Laboratories, LLC., 3011 Malibu Canyon Road, Malibu, California 90265-RL96; and Integrated Media System Center, Department of Electrical Engineering Systems, University of Southern California, Los Angeles, California 90089-2564
E-mail: wzhou@hrl.com

Asha Vellaikal

Information Sciences Lab, HRL Laboratories, LLC., 3011 Malibu Canyon Road, Malibu, California 90265-RL96
E-mail: asha@hrl.com

and

Ye Shen and C.-C Jay Kuo

Integrated Media System Center, Department of Electrical Engineering Systems, University of Southern California, Los Angeles, California 90089-2564
E-mail: yshen@sipi.usc.edu, cckuo@sipi.usc.edu

Received March 11, 1999; accepted October 9, 2000

Network-based computing is becoming an increasingly important area of research, whereby computational elements within a distributed infrastructure process/enhance the data that traverses through its path. We refer to these computations as online processing and this paper investigates scene change detection in the context of MBone-based proxies in the network. On-line processing varies from traditional offline processing schemes, where for example, the whole video scope is known as *a priori*, which allows multiple scans of the stored video files during video processing. The on-line processing system is designed to meet the requirements of real-time video multicasting over the Internet and to utilize the successful video parsing techniques available today. The proposed algorithms do scene change detection and extract key frames from video bitstreams sent through the MBone network. We study several algorithms based on histogram differences and evaluate them with respect to precision, recall, and processing latency.

Key Words: Internet video; IP multicast; MBone; video scene change detection; on-line processing, feature extraction.

[1] To whom correspondence should be addressed. Other information is available at http://www-nrg.ee.lbl.gov/vat/.

1047-3203/01 $35.00

1. INTRODUCTION

Internet packet(IP) video is emerging as an important area of multimedia application, especially with increasing bandwidths available over the network today. IP multicast is an effective means for data dissemination and sharing among a large user community. Over the past few years, real-time multimedia conferencing tools have been developed to operate over the Multicast Backbone(MBone). Unfortunately, current "multicast-aware" collaboration tools (e.g., vic [1] and vat*) only focus on effective and efficient codecs but are generally insensitive to content semantics. The semantic multicast [2] project at HRL implements a large-scale shared interaction infrastructure that provides a seamless environment for collecting, indexing, and disseminating the information produced in collaborative sessions. This infrastructure captures the interactions among users (as video or audio streams) and promotes a philosophy of filtering, archiving, and correlating collaborative sessions in user- and context-sensitive subgroupings. An important goal of semantic multicast is to create a filtering structure for making the streams of the collaborative session available to the correct users at the right amount of detail. In order to fulfill this requirement, every stream must be subjected to real-time, or near-real-time, "quick" content annotations so that the stream can be properly matched and propagated forward in the semantic multicast architecture. Annotation and summarization of the raw media content is carried out at proxies that are located in networks.

Video is an important component of the collaboration data streams. In order to enable filtering of on-line streams as well as effective retrieval of desired data from an archive, the video/audio streams need to have annotations or tags generated that describe the "content" of the stream. In its raw form, multimedia data types such as video and audio are not amenable to automated semantic interpretation and typically must be enhanced with additional information such as key words. In addition, for summarization purposes and further content extraction, the video can be tagged with scene change information as well as representative sample frames. There are some characteristics of video data for real-time content processing in proxies in the network. First, video by nature tends to be relatively larger in size as compared to other types of data. In addition, semantics extraction has traditionally been very difficult from visual data. On-line processing always places very tight time constraints that affect the complexity of the algorithms. This is especially important for data that are bandwidth intensive such as video. Thus, the operations or algorithms should clearly be conductive to real-time processing of the data. Note that content-based transcoding of the data can also be a filtering operation for assisting users who have very low bandwidth. We distinguish between on-line and off-line annotations because the issues are considerably different in the two cases. One of the biggest differences is off-line mode does not have the real-time constraint in generating the annotations. Thus off-line annotations can be more sophisticated than those generated using on-line processing. In this paper, we consider automatic on-line annotations on packet video. Automatic scene change description tags and key frames are examples of such descriptor annotations. To generate these tags, annotations based on raw video processing need to be utilized to detect whether a scene has changed and to generate a key frame that summarizes the scene. We evaluate different scene change detection algorithms w.r.t. different performance criteria on Intra-H.261 video.

This paper is organized as following: Section 1.1 provides a background of the MBone and related protocols such as RTP(real-time protocol) and MBone-related video tools. Section 2 provides related work on off-line scene change detection. Details of our system

and the different scene change detection algorithms are described in Section 3. Section 4 gives the experimental results related to our different techniques while Section 5 concludes the paper and outlines the direction of future work.

1.1. Background

The MBone is the Virtual Internet backbone for multicast IP. The MBone is layered on top of portions of the physical Internet to support routing of IP multicast packets since that function has not yet been integrated into many production routers. The network is composed of islands that can directly support IP multicast, such as multicast LANs like Ethernet, linked by virtual point-to-point links called "tunnels." The tunnel endpoints are typically workstation-class machines having operating system support for IP multicast and running the "mrouted" multicast routing daemon.

RTP [3], the real-time transport protocol, has gained widespread acceptance as the transport protocol for audio and video on the Internet. It can be used for media-on-demand as well as interactive services such as Internet telephony and provides services such as time-stamping, sequence numbering, and payload identification. The data part of RTP is a thin protocol providing support for applications with real-time properties such as continuous media (e.g., audio and video), including timing reconstruction, loss detection, security, and content identification.

Vic [1], which provides the video portion of a multimedia conference, is based on RTP. The most used compression scheme in vic is that based on "Intra-H.261," which is robust to package loss. Intra-H.261 is a small subset of H.261 in that: (a) only intramode frame in H.261 is adopted and (b) no prediction error computation is involved—i.e., no inverse quantization and IDCT. Intra-H.261 provides improvements in compression gain and substantial improvement in run-time performances. Traditional compression algorithms are designed for constant bitrate channels, but the Internet is a harsh environment for compressed video signals because it has relatively high packet loss rate, and traditional compression algorithms like MPEG and H.261 cannot do very well in this environment. The reason is that these algorithms use motion-compensating prediction to remove temporal redundancy. Although resynchronization can be obtained using intracoded frames, the resynchronization interval is usually tens or hundreds of frames, which makes the possibility of an error-free interval very low, making it virtually impossible for high-quality video transmission. One solution to this problem is through "conditional replenishment" in which motion compensation is kept. In this method, each frame is partitioned into small blocks and information about only those blocks that change beyond some threshold is transmitted. Updates are always intracoded to avoid error propagation in prediction. To decide whether to encode and transmit a block, the conditional replenishment algorithm computes a distance between the reference block and the current block. As is the standard practice with common motion-compensation algorithms, vic runs conditional replenishment exclusively on the luminance component of the video. To avoid background noise and motion artifacts, an absolute sum of differences rather than a sum of absolute differences is used. If the block of reference pixels is $(r_1, r_2, \ldots, r_n)$, the block of new pixels is $(x_1, x_2, \ldots, x_n)$, and the threshold is T, then the new block is selected if

$$\left| \sum_{k=1}^{n} (r_k - x_k) \right| > T. \tag{1}$$

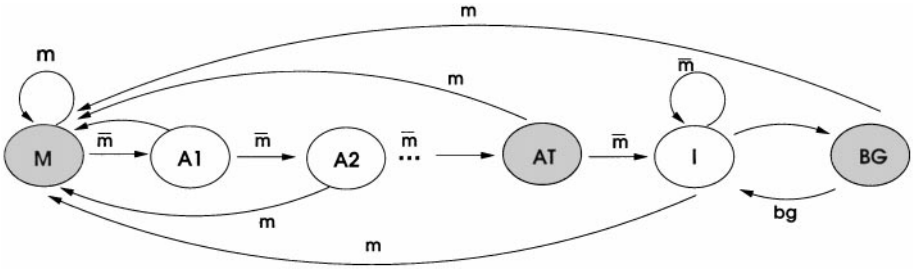

FIG. 1. Block aging algorithm. A separate finite-state machine is maintained for each block in the image. State transitions are based on the presence (M) or absence (M-) of motion within the block. The block is replenished in the shaded states.

A background processing continuously refreshes all the blocks in the image to guarantee that lost blocks are eventually retransmitted. This solution works well for video conferencing for three reasons: First, blocks that need to be transmitted usually contain the motion portion. Due to the spatial locality of the motion, they will be transmitted again even if a packet loss occurred previously. Secondly a typical scene in a video conference is a small motion portion in large static backgrounds, making the conditional replenishment the suitable choice. Thirdly, the computational complexity for encoder is reduced.

The above block selection algorithm will cause noticeable blocking artifacts. This is because the block selection hysteresis may take hold and some blocks may no longer get replenished even though the block continues to change. Hence, the final block has a persistent error with respect to the final static state. The problem is solved by the conditional replenishment algorithm. When the selection algorithm ceases to sent a given block, that block is aged and resent at some later time. This algorithm can be illustrated in Fig. 1. Each block in the image has a separate finite state machine and a block is encoded and transmitted only when it is in the shaded states. Whenever the block selection algorithm detects motion in a block, the state machine transitions to the motion state (labeled M). At the age threshold (state A_T), a block is sent and in turn enters the idle state (I). Vic fixes A_T at 31. Besides, a fill process, which is a background process running to continuously refresh all the blocks in the image to guarantee that lost blocks are eventually retransmitted, selects some number of idle blocks in each frame and spontaneously transitions them to the background state (BG).

2. RELATED WORK

Shot boundary detection typically represents the first step in structuring video sequences for content-based retrieval. Several techniques have been studied as discussed below for shot boundary detection, including pixel differences, statistical differences, histogram comparisons, and edge differences.

There are several schemes based on pixel value comparisons between adjacent frames in a video. Zhang *et al.* [4] used a pixel and block comparison method on uncompressed bitstreams. In this scheme, two frames are significantly different if the number of pixels that change in intensity values more than a given threshold exceeds a certain percentage of the total number of pixels. The disadvantages of this method is that it is not robust and is extremely slow, and manually adjusting the threshold is not practical. Shahraray [5] divided

the frame into 12 nonoverlapping blocks and found the best matching from neighboring frames (same as in motion estimation). The weighted sum of each block difference provides the comparison measurement. Again, this method is quite slow and thus not suitable for on-line scene detection. Hampapur *et al.* [6] divided the change in gray level of each pixel between two images by the gray level of the pixels in the second image. This method is very sensitive to camera and object motion. Kasturi and Jain [7] used likelihood ratio comparison and computed the mean and standard deviation of the gray levels in different regions of the image. This method is better in noise tolerance, but is computationally expensive and also have a high false alarm rate.

Histogram comparison is the most commonly used method in shot detection. Histograms of frames are compared to detect an overthreshold difference. The difference of two histograms is calculated as the summation of all absolute difference between the two histograms on each gray or color level. Nagasaka and Tanaka [8] compared several simple statistics based on gray level and color histogram and found that the higher order of histogram comparison may not be more efficient than the gray and color histogram comparison techniques. Zhang *et al.* [4] proposed a histogram twin-comparison approach for detecting gradual transitions like wipes and dissolves. This method requires two cutoff thresholds: a higher threshold (T_i) for detecting abrupt transitions and a lower one (T_l) for gradual transitions. Also this method requires two stages in order to detect gradual transitions.

Several compressed domain shot detection methods have used techniques based on the DCT coefficient values. Arman *et al.* [9] used differences in the DCT coefficients of JPEG-compressed frames as a measure of frame similarity. They sample the frames temporally to speedup the searching of a boundary. Potential boundaries are checked using the color histogram difference method. Shen and Delp [10] presented an algorithm to reconstruct DC coefficient images of a DCT and motion-compensation compressed video sequence, e.g., MPEG. Yeo and Liu [11] also use the DC image, a form of reduced images, for various scene analysis tasks. The DC image is derived from DCT-based and motion-compensated-DCT-based compression techniques such as motion JPEG and MPEG. The histograms of the dc coefficient images can be used to detect scene changes. In their algorithm, they tried to detect dissolving scene changes using average values of past 20–30 frame histograms, which is not possible for our on-line scene change detection algorithm because it is computationally complex and not suited for on-line processing.

As mentioned earler, we focus on shot detection in an on-line environment. Since such processing puts stresses on real-time processing, we need an algorithm that is simple and straightforward without too much memory or buffer requirements. While off-line scene change detection is now a pretty mature research area, we extend these work to the on-line scenario in this paper.

3. SYSTEM ARCHITECTURE

Currently, we have developed a scheme that summarizes real-time video in terms of a few key frames that are representative of that video. The main component of this scheme is based on real-time scene change detection of RTP Intra-H.261 compressed video stream. Here a video is split into meaningful scenes such that each scene corresponds to a shot that is different from other shots based on a particular criterion. A video can then be summarized utilizing the different scenes that make up the whole stream.

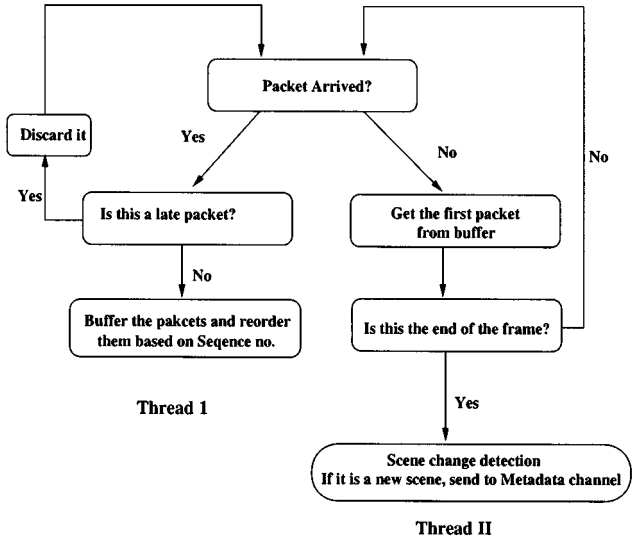

FIG. 2. On-line scene change detection of H.261 video.

The processing is shown in Fig. 2. As shown the agent takes in a real-time video stream and buffers the packets. The packets in buffer are then partially or fully decompressed depending upon the scene change detection algorithm. In our system, we studied a few different algorithms, including an algorithm that utilizes full decompression to get full frames for comparison and another one that uses partial decompression to get information of changed blocks so as to estimate the extent to which the full frame has changed since the last frame. The scene-change detection is based on histogram comparisons between adjacent frames of the video stream. We studied different algorithms and evaluated their performance by comparing their precision, recall rate, and processing latency. Once a scene change is detected, a key frame is generated from that stream and sent out as a scene in a multicast channel.

The major design goals for our proposed algorithm are considered here. We based our work on Intra-H.261 video over RTP protocol. As mentioned previously, the Intra-H.261 algorithm only encodes the differences of macroblocks of the later frames from previous frames to get the most temporal compression rate. Three major goals that influenced our algorithm design are:

- On-line processing speed,
- Precision rate and recall rate, and
- Packet buffering and synchronization.

A major concern for our algorithm design is the scene change detection precision and recall rate. To save bandwidth for our on-line annotation, we only send out key frames for further processing, so accurate scene change detection is very important as we do not want to waste our limited network and processing resources on the nonkey frames. Generally there is a tradeoff between precision and recall rate. Our ultimate goal is to achieve the highest recall rate while still keeping the precision as high as possible. Recall rate is important for our on-line annotation and filtering proxies because the key frames are the fundamental elements for further annotation and filtering. In an on-line scenario,

packets for video frames need buffering and synchronization. Intra-H.261 has the advantage over prediction-intensive codecs such as MPEG in that the packets of Intra-H.261 are all independent of each other, whereby the effect of any loss of packets will be limited to a few frames, and therefore has much better tolerance to packet losses than MPEG. Unfortunately the compression and loss tolerance advantages of Intra-H.261 come at the cost of increased computational complexity due to the increased number of intracoded blocks because the blocks that could be coded with motion vectors in H.261 must be fully coded in Intra-H261. For the same reason, the decoding speed of Intra-H.261 should be much slower than MPEG decoding too. So speed of video processing algorithms for Intra-H.261 is very essential to support real-time video summarization and annotation in on-line application. In our implementation, we used a linked list to buffer packets as many as necessary, but our scene change detection algorithm should be designed to be processed as fast as possible such that it can keep the buffer as small as possible to keep up with the original video flow speed.

3.1. Scene Change Detection Algorithms and Implementation

The basic framework of our proxies is the following: our video scene detection proxy opens two sockets to the specific multicast addresses/ports. On the arrival of a packet, it is buffered for processing. Buffering is implemented using a linked list, ordered by the sequence number of each packet. Thus packets that arrived out of order are taken care of during the buffering. The link-list buffer also keeps track of the frame that has just been processed. If a packet for that frame arrives after the frame has been processed, it will be discarded. In the meanwhile, to speed up the decoding and scene change detection, we keep packet buffering and packet processing in parallel by using multithreading programing in POSIX.1c.

Packet buffering includes packet copying/synchronization, while packet processing includes decompression/partial decompression, frame boundary detection/frame assembly, and scene change detection. When the packets are not always arriving in order, we must provide a mechanism to synchronize the frame without suffering too much on the processing latency. Packets are processed in an ordered manner from the buffer. We always check the Frame Marker Bit in the RTP header to see if a whole frame has arrived. If the last packet in that frame has arrived, we start with the scene change detection algorithm. In the meanwhile, we reset the time stamp for a new frame. If a frame has a few packets, then the first few packets of the frame generally must stay at buffer longer than the last few packets. If a packet belongs to an old frame that has been processed already, that packet is discarded immediately. This explicit resynchronization of frame packet is due to the property of Intra-H.261 framing protocol [1]: packets in Intra-H.261 are independent of each other and can be decoded in isolation or in arbitrary order up to a frame boundary. To satisfy our real-time scene change detection requirement, we have implemented several scene change detection algorithms by combining off-line scene change detection algorithms [4] and coding features in the Intra-H.261 codec [1]. Results of a detailed study and comparison of these algorithms are provided in Section 3.3. The details of various scene detection algorithms we implemented are described below:

1. Full frame scene changed detection (FDY and FDYUV). In this first implementation, we fully decompress every packet for the whole frame and only use the luminance values to get a 256-bin gray-scale histogram over the entire frame. The dissimilarity of images measured by luminance histogram is decided by the following: given two histograms, H_i

and H_j, each containing n bins, the normalized match index([0, 1]) of the intersection of histograms is defined as

$$S(H_i, H_j) = \frac{\sum_{k=1}^{n} \min(H_i, H_j)}{\sum_{k=1}^{n} H_i}. \quad (2)$$

It is easy to see that $\sum_{k=1}^{n} H_i$ should be the total number of pixels of the frame. For our purpose, $d_c(H_i, H_j) = 1 - S(H_i, H_j)$ is the dissimilarity index to indicate dissimilarity between the two images. We call this algorithm FDY (full decompression with luminance histogram scene change detection algorithm).

Besides using only the luminance histogram, we also implemented a color histogram scene change detection algorithm by taking both luminance and chrominance information into consideration for a color histogram. This variation of the first method is called FDYUV (luminance and chrominance histogram scene change detection method by using full-frame decompression). The FDYUD is almost the same as FDY. The only difference is that we consider color information by quantizing YUV into one byte with the bit ratio of Y : U : V = 2 : 3 : 3, which are 4, 8, 8 levels for Y, U, V respectively. The reason for this bit assignment is that by assigning less bits to Y than to U and V, we can effectively reduce the false alarms caused by illumination changes, which occur frequently, especially in news video because of frequent flashing light. Besides, by putting more weight on the requantization of U and V, we are able to test the effect of chrominance information on the scene change detection. Based on our experiments, this quantization of YUV is found to be adequate.

2. Frame sampling (FSDY and FSDYUV). In this second type of implementation, we used frame sampling to speed up the processing procedure. Here we refer to the algorithm that uses frame sampling together with histogram comparison as FSDY if the histogram used in the histogram scene change detection algorithm only contains luminance information, and FSDYUV if the color histogram contains all YUV information. We observed that generally shots with same/similar scenes would last at least 2 to 3 consecutive frames. So we choose the frame sampling frequencies of 2 frames/detection. The histogram scene change detection is the same as that in the type-1 algorithm.

3. Partial decompression algorithms (PDY and PDYUV). This algorithm is different from general scene change detection algorithms in the compressed domain as we take advantage of representational structure present in compressed video formats [9, 12]. Also we are constrained to standard streaming protocols for video on the Internet (RTP [3]). As mentioned earlier, Intra-H.261 only has an intra mode, so it does not contain any motion vector. Nor does it always send whole compressed data in a frame, but from the Intra-H.26l encoder's header, we are able to know the position and the total number of changed marcoblocks. However, this macroblock information is still too limited for us to judge whether it is a scene change detection, because macroblocks in a frame are sent when any of the following three reasons are satisfied. First, motion is caused by changes of macroblock content. Intra-H.261 only has an intra mode and there are no motion vectors for each macroblocks. If there is a lot of object motion, most of the macroblocks in the frame are going to be changed locally and all of them must be encoded and sent out. Second, packet loss from the network and packet dropping from packet buffering will cause the number of changed macroblocks to not be accurate. Third, refreshments due to the conditional compensation algorithm make macroblocks received in that frame to not be always different

from those of the previous frame. Intra-H.261 uses the conditional compensation algorithm to make up the quality losses caused by packet losses from network and aging blocks. From Fig. 1 we can see that the conditional compensation algorithm is randomly refreshing those aging blocks and thus adds uncertainty to the macroblock numbers. From the above discussion, although we are not sure that all those received macroblocks are new with respect to the previous frame, we have observed that there is generally no scene change if only small portions (small number of macroblocks) of a frame are changed and encoded. So for those frames with few new macroblocks, we can save processing time by not doing full decompression and histogram scene changes, which are generally computationally expensive. Only for those frames with a large number of changed macroblocks will we use only either luminance histogram scene Change detection (here we call it PDY) or YUV histogram scene change detection (PDYUV) to do new scene detections.

4. Mixture method (MY and MYUV). Finally, we integrated the frame sampling algorithm, partially decompression, and the frame comparisons algorithm with the histogram scene change detection algorithm. We tested frames with sampling techniques used in FSDY and FSDYUV and did scene change detection only when it was necessary; in other words, we decompressed the frame and compared it with previous decompressed frames using histogramscene change detection techniques only when the percentage of changed macroblocks fell in ranges from 20 to 80%. For frames with a small percentage of macroblock changes, we treated it as no scene change while for a large percentage of macroblock changes, we treated it as a new scene directly, the reasons for this being pretty much as the same as those discussed in previous paragraphs. We refer to this integrated method with Y histogram comparisons for those necessary frames as MY, and that with YUV histogram as MYUV.

4. EXPERIMENTAL RESULTS AND DISCUSSION

4.1. Results for Algorithms

In our experiments, we encoded a 10-min. clip from CNN Headline News at different bit rates ranging from 64 kbps to 1 Mbps. The processing was carried out on an Ultra-2 Sun workstation with 256 MB RAM. Video was multicasted into our local LAN with RTP protocol. Our local network has bandwidth up to 100 Mbps, which should be enough bandwidth for our maximum video bandwidth requirement in our experiment. This paper does not consider packet losses caused by the network.

The performance was measured on following three criteria: processing latency of packets, precision, and recall rate for videos with different bitrates and with different scene change detection algorithms. Recall and precision rate of an algorithm are defined as:

- recall = correct detects/(correct detects + missed detects)
- precision = correct detects/(correct detects + false alarms).

Here “correct detects” are the number of true scene changes detected by our algorithm, “missed detects” are the scene changes in the video that are not detected by our algorithm. “False alarms” are those video frames that are not new scene frames but detected as new scenes by the algorithm. The precision and recall rate measurements of different algorithms at different bandwidths are listed in Table 1. The threshold used in algorithms described in Section 3.1 was determined experimentally to ensure a desired level of performance for all news videos.

TABLE 1
Precision and Recall Rate of Scene Change Detection Algorithms

Algorithms	Precision	Recall rate	Precision	Recall rate	Precision	Recall rate	Precision	Recall rate
FDYUV	97%	90%	97%	90%	97%	90%	97%	90%
FDY	90%	98%	90%	98%	90%	98%	90%	98%
FSDYUV	96%	76%	96%	77%	97%	80%	97%	82%
FSDY	83%	87%	84%	89%	86%	90%	86%	90%
PDYUV	98%	87%	98%	87%	98%	87%	98%	87%
PDY	86%	95%	86%	95%	86%	95%	86%	95%
MYUV	55%	99%	58%	98%	60%	98%	60%	98%
MY	50%	99%	52%	98%	55%	98%	60%	98%

Latency time is generally used to measure the processing speed of the algorithm for real-time applications. Here we defined the latency time of our scene change detection algorithms as the average buffering time of last packet in each frame. For the last packet of every frame, we designate the time right after our processor receives that packet from the network as the starting buffering time of the packet, and we identify the time when our processor sends out the packet to the metadata channel as the stopping buffering time. The average buffering time is the total buffering time for the last packets of all frames averaged over the frames. The reason for us to define the latency time this way is illustrated in Fig. 3.

Figure 3 shows the packet buffering mechanism. Packets are buffered until each frame is finished with processing. The main processing for each frame is the total packet decompression time for all packets in that frame and the scene change detection time. Assume that there are m_1 packets in the nth frame, with M being the total number of packets in the buffer. Also assume that the average decompression time for each packet is about TD and that the average for scene change detection time for each frame is TS. Since the typical maximum packet size is limited to 1024 bytes, we can approximate the arriving rate r (packet/second) of packets based on the data bitrate and thus we know that about every $\mathrm{TR} = 1/r$ s, the buffer will get one packet. Consider the following queueing conditions in the stored buffer:

- When packets are queueing up in the buffer, M is bigger than 0. Then the buffering time for the mth packet in nth frame $P_{n,m}$ is approximated as $\mathrm{BT}_{p,n,m} = (M - m_1 + (m-1)) \times \mathrm{TD} + n \times \mathrm{TS}$. So within the same frame, the first few packets in the frame must stay longer in the buffer than the last few packets.
- When the buffer always contains only one frame data, i.e., $M - m_1 = 0$, the packet arrival rate is about the same as the packet processing rate. So the buffering time for the packet in frame n is given by $\mathrm{BT}_{p,n,m} = (m_1 - (m-1)) \times \mathrm{TD} + \mathrm{TS}$. Similarly, the buffering time for the m_1th packet, i.e., the last packet, of the frame is $\mathrm{BT}_{p,n,m} = \mathrm{TD} + \mathrm{TS}$.

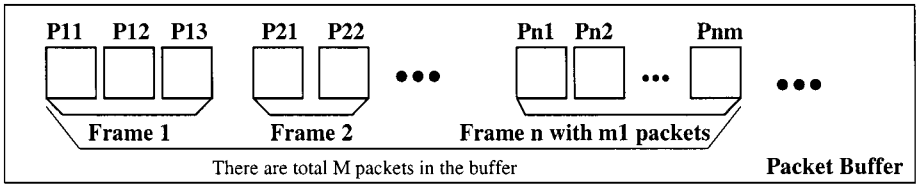

FIG. 3. Packet buffering mechanism in our algorithm.

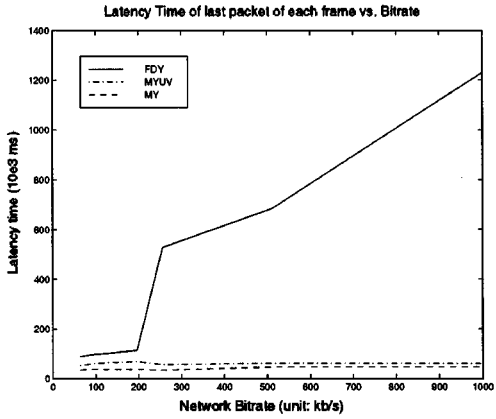

FIG. 4. Bit rate affects processing latency.

Packet processing includes packet buffering/packet synchronization, decompression/ partial decompression, frame boundary detection/frame composition, and scene change detection. The synchronization time of a frame generally requires that early packets must wait for the last packet before the frame can be checked out.

• In the case where the bit rate is very low, often the buffer is in an incomplete state with respect to a full frame. Thus, the nth frame is not complete in the buffer, i.e., $M - m_1 < 0$ and $m < m_1$. In this case, the buffer time of packets in the frame is $\mathrm{BT}_{p,n,m} = (m_1 - (m-1)) \times \mathrm{TD} + (m_1 - m) \times (\mathrm{TR} - \mathrm{TD}) + \mathrm{TS}$. However, the buffer time of the m_1th packet (i.e., the last Packet) of the frame is $\mathrm{BT}_{p,n,m} = \mathrm{TD} + \mathrm{TS}$.

From the above discussion, we can see that the buffer time of the last packet of each frame depends on the decompression time and the scene change detection speed, but does not depend on the bitrate. While the buffering of the packets other than the last packets depends on all of the parameters of TD, TS, and TR, this cannot truely reflect the latency time caused by our video scene change detection processing. This is also very clearly shown in Fig. 4 and Fig. 5. From Fig. 4, we see that the latency time caused by the video processing should be the same for the last packet for all of the frames when video processing is fast

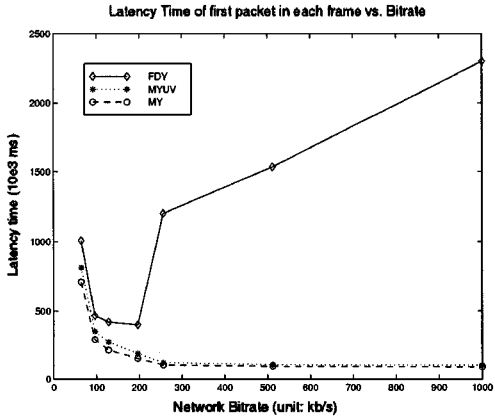

FIG. 5. Bit rate affects processing latency.

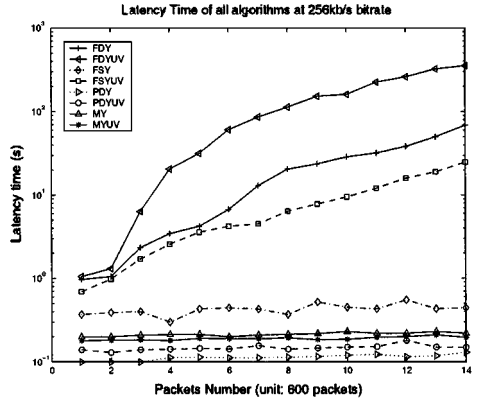

FIG. 6. Latency of video processing.

enough. Typically the early packets in a frame generally stay longer in the buffer than the last packet in the frame. When the bandwidth is too low and our processing is faster than the media flow, the earlier packets in a frame will have to wait for the later packet in the frame with the proxy processor in an idle state.

Figure 6 shows the latency time vs packet number. As we discussed before, if the scene change detecting algorithm is fast enough to keep pace with the media flow, then every frame should have about the same processing latency. If this is not the case, then the later frames must wait in the buffer and thus will have longer latency time than earlier frames. Also, we measured output data flow rate at different input data flow rates; the results are shown in Fig. 7.

4.2. Discussion

In this section, we are going to discuss the results we got from the experiment. The major issue is whether the algorithms meet with our design goals.

First, we compared precision and recall rate among all algorithms on four data sets with different bitrates. From Table 1, we observed that the histogram scene change detection

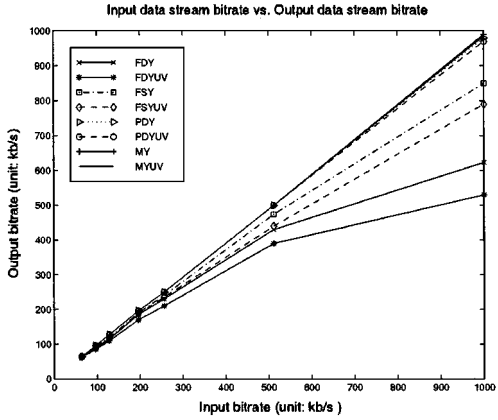

FIG. 7. Output flow vs input flow.

algorithms are very consistent and the color histogram scene change detection algorithms are more accurate than luminance histogram scene change detection algorithms since the former have a higher precision rate than the latter. This is because algorithms with only luminance are not sensitive to color changes, which in many cases represent object/background changes. Also when only luminance intensity changes, such as "lighting," it would cause false alarms and thus decrease the precision. Among all the algorithms, those running on full decompressed frames have higher precision.

Among algorithms, MY and MYUV are the two with worst precision. However they have the highest recall rate. There are quite a few reasons to contribute to this: first, in these two algorithms, most scene change detections are done without histogram comparison. Frames with a lot of motions are detected as key frames because the algorithm senses a lot of changed macroblocks, even though these changes only correspond to macroblocks shifting between adjacent frames due to object motion. This problem also arises in scenes that are involved in a gradual transition-like dissolve. As discussed in the background section, another reason for misdetection is that the conditional compensation algorithm will refresh aged macroblocks randomly and as a result this refreshment will increase the number of macroblocks and cause false alarm. In summary, the advantage of this mixed method is that the processing speed can be improved to some extent since we only perform partial decompression along with as less as possible histogram comparisons. Besides, when the recall rate is the more concerned performance measurement, this method will make a good candidate.

Frame sampling can speed up processing and shorten processing latency. This algorithm has worse precision because the Intra-H.261 codec only sends out macroblocks that are different from the previous frame and thus any bypass of video frame has the possibility of bypassing a scene change, especially in cases where the following frames right after the bypassed frame only have small motion. The comparison of the latter with a previous scene may be below threshold. In this case, we will miss some key frames that affect the precision, but we also see that precision and recall rate for frame-sampling algorithms are getting-better when the bit rate is higher. This is because at higher bit rate, video frames are more redundant and thus key frames are not easily missed.

On the other hand, recall rate is a tradeoff to algorithm precision. Generally luminance histogram scene change detection algorithms have higher recall rate, because they are using a looser criterion than YUV histogram. FDY is the best algorithm when recall is important, while FDYUV is the best with precision and recall rate, but these two algorithms have the disadvantage for real-time scene change detection because they have much longer processing latency than other algorithms. Things get worse when the video is longer and encoded with higher bitrate. After a certain point, the processing latency will exceed the tolerance. Keep in mind that when we are detecting new scenes for on-line video, processing speed and recall rate are normally more important than precision. The importance of processing speed is very obvious. High recall rate is very important to keep video fast on-line annotation and summarization without losing too much information.

Another most important performance measurement in the real-time processing algorithms is real-time video processing latency. In the video processing, generally inverse DCT and color requantization are the two most computationally expensive parts. The latency time measurement of frames are measured and shown in Fig. 6. From the graph, we can see that color histogram generally takes longer time than luminance histogram comparison because

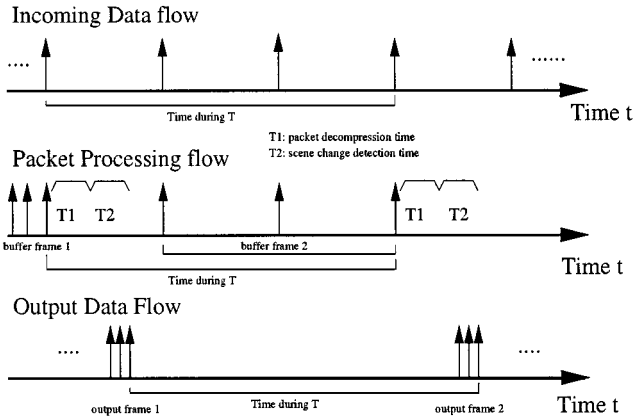

FIG. 8. Data flow without video processing delay.

we must spend a relatively bigger chunk of the processing time on color requantization. From the above discussion, we conclude that algorithms with luminance only should always be considered first whenever the processing speed becomes the bottleneck. Our experiments show that all of the algorithms except FDYUV, FDY, and FSYUV are good for real-time processing with respect to the processing speed. They can keep pace with video flows ranging from 64 to 512 kb/s, which is the common bandwidth of packet video over the MBone. Figure 6 also shows that the latency time for all packets within the video using algorithms other than the above three are flat when the bit rate is at 512 kb/s, which means the algorithms can keep pace with media flow, but this is not the case for FDYUV, FDY, and FSYUV. The graph shows that the later frames in a video data sequence must wait a longer time to be processed because the processor is so slow that it causes the packets to queue up.

We also measured output data flow rate with different input data flow rates; if the system is real time, then the output data flow rate should be the same as the input data flow, as shown in Fig. 8 and Fig. 9. From Fig. 7, we know that our fast algorithms (FSY, PDY, PDYUV,

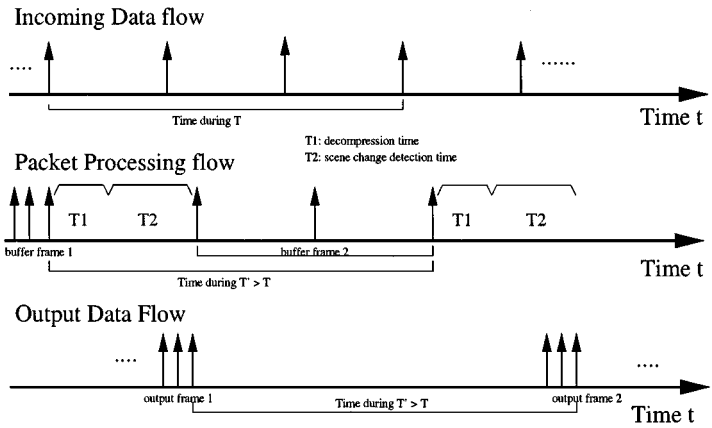

FIG. 9. Data flow with video processing delay.

MY, and MYUV) can keep pace with input video streams upto 1 Mb/s, while FSYUV, FDY, and FDYUV can only keep data flows upto 256 Kb/s.

5. CONCLUSION

We presented a few effective methods of on-line video scene change detection over MBone video for the purposes of annotation/filtering. Detailed study on these algorithms with respect to precision, recall rate, and latency are given. One of the main advantages of our algorithms is their ability to support real-time video processing on the network. Joint algorithms based on video codec characteristics for acquiring fast and accurate scene detection were carried out. Both global color histograms were extracted and block information of DCT coding was used. Performance of algorithms running with different bit rates was also studied. We presented results that proved that our algorithms are capable of satisfying on-line annotation needs. Also with the detailed performance studies of these algorithms on different data bandwidths, we are able to choose the right algorithm with the best performance in each case. However, since our algorithms are mainly targeting real-time video scene analysis, improving processing speed and reducing latency are two more issues of concern, which makes it improper to compare our algorithms with other off-line scene detection techniques, such as those we mentioned in the Related Work section, where video data are processed off-line.

In the future, we plan to study more video on-line annotations based on video content, such as key frame classification. Because of the complicated nature of video processing, we plan on using multiple workstations working in parallel to realize more sophisticated real-time annotation. We also plan to study algorithms that are tolerant of packet losses in the network. On the other hand, just as the vic tool adapts the video quality to the bandwidth, we are going to investigate algorithms of on-line annotation and filtering that are bandwidth adaptive.

ACKNOWLEDGMENT

The work presented in this paper is supported by the DARPA project on Semantic Multicast, N66001-97-2-8901.

REFERENCES

1. S. McCanne and V. Jacobson, Vic: A flexible framework for packet video, in *Proceedings 4th ACM International Conference on Multimedia*, 1995.
2. S. Dao, E. Shek, A. Vellaikal, R. Muntz, L. Zhang, and M. Potkonjak, Semantic multicast: Intelligently sharing collaborative sessions, *J. ACM Comput. Surv.* 1999.
3. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A transport protocol for real-time applications, in *RFC 1889*, Jan. 1997.
4. H. J. Zhang, A. Kankanhalli, and S. W. Smoliar, Automatic partitioning of full-motion video, *ACM/Springer Multimedia Syst.* **1**, 1993, 10–28.
5. B. Shahraray, Scene change detection and content-based sampling of video sequences, *Proc. IS&T/SPIE* **2419**, 1995.
6. A. Hampapur, R. Jain, and T. Weymouth, Digital video segmentation, in *Proceedings 2nd ACM International Conference on Multimedia*, pp. 357–364, 1993.
7. R. Kasturi and R. Jain, Dynamic vision, in *Computer Vision: Principles* (R. Kasturi and R. Jain, Eds.), pp. 469–480, IEEE Computer Society Press, Washington, DC, 1991.

8. A. Nagasaka and Y. Tanaka, Automatic video indexing and full-video search for object appearances, in *Visual Database System II* (E. Knuth and L. M. Wegner, Eds.), pp. 113–127, Elsevier, Amsterdam, 1992.
9. F. Arman, A. Hsu, and M.-Y. Chiu, Image processing on compressed data for large video databases, in *Proceedings 1st ACM International Conference on Multimedia*, pp. 267–272, 1993.
10. K. Shen and E. J. Delp, A fast algorithm for video parsing using mpeg compressed sequence, *Proc. IEEE*, 1995, 252–255.
11. B.-L. Yeo and B. Liu, Rapid scene analysis on compressed video, *IEEE Trans. Circuit Syst. Video Technol.* **5**, 1995, 533–544.
12. H. J. Zhang, C. Y. Low, and S. W. Smoliar, Video parsing and browsing using compressed data, *Multimedia Tools Appl.* **1**, 1995, 89–111.