ARCHITECTURE OVERVIEW OF MOTION VECTOR REUSE MECHANISM IN MPEG-2 TRANSCODING

Javed I. Khan, Darsan Patel, Wansik Oh, Seung-su Yang,
Oleg Komogortsev, and Qiong Gu

Technical Report TR2001-01-01

Internetworking and Media Communications Research Laboratories
Department of Math & Computer Science
Kent State University, 233 MSB, Kent, OH 44242

(Last Revised January 31, 2001)
javed@kent.edu

1 OVERVIEW

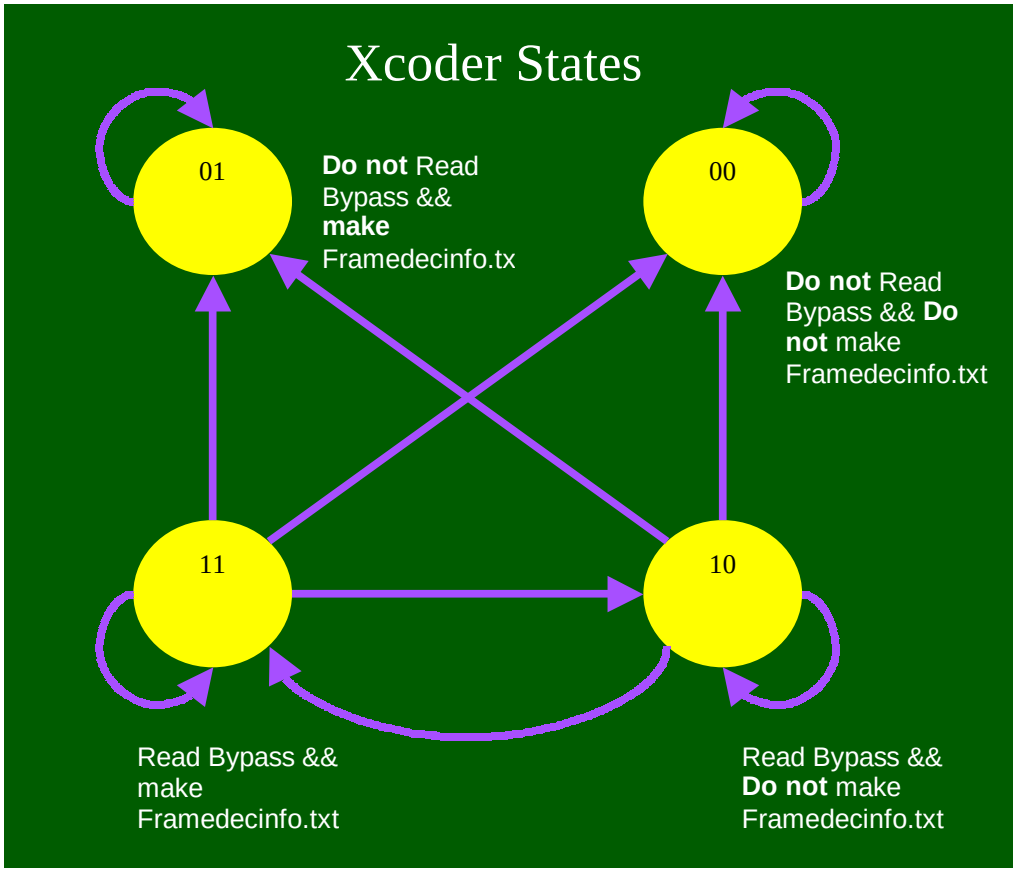
In a typical large network (like the global internet) links in the path from the source to the sink have various capacities. The individual link capacities also are not be static. These links are shared, thus are subject to transient congestion. Also, the network elements switches, routers) available over the network themselves have unequal computational power, and these too are shared. Thus, the computational power allocated to individual adaptation process is also dynamically variable. This second level of asymmetry is particularly significant for the programmable networks—such as active networks [5,6]. In this research we take a fresh look into this problem from a unified perspective of adaptation with respect to both the fundamental network resources—bandwidth and processing power.

The transcoder itself offers the first level of adaptation [4,7]. The transcoder has the dynamic control to modify the outgoing bit rate. It can be deployed as an active net capsule in side the network. However, video transcoding is a computationally daunting task by its own virtue [4,7]. It becomes more so when we plan to perform it in-stream in real time inside on a network. Current state-of-the art in video compression technology requires custom chipset to obtain real-time performance in MPEG-2 encoding. Roughly speaking, something close to CIF video (CCIR 4:2:0 CIF=352x240 at 30 frames/second or 352 x288 at 25 frames/second) can be decoded in software satisfactorily. It requires processing of 69,300 blocks per second. In comparison, a broadcast quality video (CCIR-601 4:2:2: 720x480 at 30 frames/second or 352x576 at 25 frames/second) requires processing of 405,000 blocks per second, while a production or medical quality video (MPEG-2 HIGH@HIGH-1440 profile=1920x1152x 60 frames/second) will require processing of 5,184,000 blocks per second. Active network [1,5,6] based transcoding adds further computational

challenge to the above scenario. First of all, a Transcoder is composed of a decoder and an encoder. Thus, it is computationally a more complex task. Secondly, on an active network the available processors are expected to be general purpose with general programmability. Which almost excludes the possibility of using any custom chip set (although field programmable logic may still be an option). Finally, an active network processor, by definition and location, is going to be a highly shared resource. The competing processes will also be real time processes. Thus, not only its processing performance, but also the complexity of the Transcoder itself has to be highly optimized.

In this research, as a fast step we have investigated quite a few techniques for accelerated transcoding. Rather than their straight implementation, however, what is more important is to note that, like any approximation algorithm—the gain in speed generally does not come free. It is a three-way tradeoff between the *speed* of the task, gained rate *compression,* and the *quality* in the original transcoding task. Therefore, on the asymmetric node capacity network, we demonstrate an self-organizing transcoder, where the very transcoder can operate at multiple states, and automatically reconfigure itself with multiple compute resource footprint versions. These varying configurations offers multiple choice points in compute resource plane—such as CPU cycle requirement, memory foot-print of data segments and instruction buffer and CPU overhead.

This document outlines a system called **self-organizing active transcoder** that we have implemented for demonstrating this concept. In this concept implementation we have chosen to implement a bi-state transcoder that offers choice in the computational speed. In MPEG-2 transcoding, motion vector estimation [2,8] is the single-most major computation component consuming as much as about 25-80% of total transcoding cycles. In transcoding we developed a novel technique where motion vectors can be reused in the recoded stream directly from the



encoded stream. It can thus significantly reduce the active-cycle consumption by the transcoding.

The transcoder can operate in 4 states. In addition to motion vector bypass / full logic mode, it can also choose between a dynamic and non-dynamic states. Once, the transcoder moves into non-dynamic state, its mode becomes final and cannot be changed. This transcoding system is intended for processing stream inside a programmable network such as active network.

The Figure above shows the state transition diagram. The engineering of this complex system has been documented in a series of technical documents. This document only outlines the architectural detail involved in the implementation of the motion vector bypass/reuse mechanism.

1.1 What is motion vector bypass?

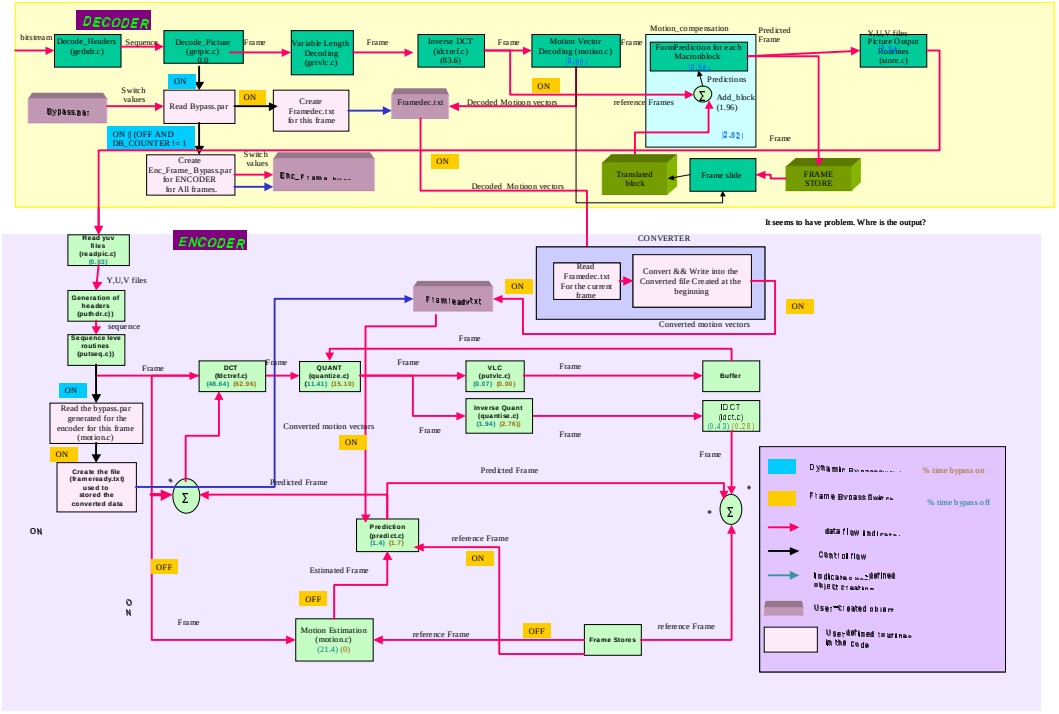
When the encoder processes data that takes heavy processing time in motion vector estimation. The motion vector bypass is how we can reduce processing time in encoder by reusing the motion vectors extracted by the decoder from the bitstream and plugging it into the encoder instead of estimating the motion vectors in the ENCODER .

1.1 What is the Dynamic Motion Vector Bypass?

The idea of Dynamic Motion Vector Bypass refers to the ability of switching or off the motion vector bypass per frame basis during the run-time of the transcoder (in other words dynamically switching ON and OFF motion vector bypass per frame basis.)

1.2 What is the problem for reusing motion vector?

When we are reusing the motion vectors or in other words extracting the motion vectors from the decoder) we do not get the optimum motion vectors because during the original encoding (i.e



before decoding into the transcoder) into MPEG format some data is lost due to compression. Therefore ENCODER OF THE TRANSCODER (or we can call it the REENCODER restimates the motion_vectors using the motion_estimation() routine to compute optimum motion vectors so as to get the optimum quality of the video. Hence we need to convert the motion vectors extracted from the decoder to match their optimum values.

2 SYSTEM ARCHITECTURE

The diagram above shows the architecture of the system. The notations (also shown in the bottom right box) are explained below.

- First notation in the diagram is indicating the status of the Frame bypass switch with ON or OFF written on it and this symbol is found on many data links in the diagram indicating what the flow of the data when it is OFF and when it is ON.
- Second notation in the diagram is indicating the status of the Dynamic bypass switch with ON or OFF written on it and this symbol is found on many data links in the diagram indicating what the flow of the data when it is OFF and when it is ON.
- Third notation is indicating that the links with that color is the data links in the diagram.
- Fourth notation is indicating all the links which carry control information in the architecture.
- .Fifth notation is to identify the links in the architecture where the medianet_incorporated objects are created and when .
- Sixth notation is to identify the medianet_created data elements in the architecture.
- The Seventh notation is used to identify the routines regarding the motion vector bypass incorporated by medianet in the original source code .
- The eighth notation is indicating the percentage of total running time for that particular routine when the FRAME_BYPASS_FLAG _SWITCH is OFF.
- The ninth notation is indicating the percentage of total running time for that particular routine when the FRAME_BYPASS_FLAG _SWITCH is ON.

2 FUNCTIONAL OVERVIEW

2.1 Decoder's process:

Initially the header information is extracted from the incoming bitstream by the decoder routine "Decode header". The stream of mpeg-2 is encoded in layers such as sequence, GOP, pictures or frames in GOP. Hence the next routine decodes the frame level information. Then a VLC table look up and inverse DCT is applied to each frame. This process gives various information along with the motion vector information. If the **frame bypass flag is set** then **at this point** of the decoding process the **motion vectors for each macroblock of each frame is trapped and stored into a set of user-defined file series (one file for each frame) called Vector Files**. The decoded motion vectors are generally not directly usable in a new encoding. Therefore, each vector file is then processed through a **CONVERTER logic designed at the lab.** The CONVERTER is designed to convert the invalid values of the motion vector data from the decoder to the valid values. Invalid values are obtained at decoder because of loss of certain data due to compression in

the original encoding of the bitstream. The details of the converting routines of the CONVERTER will be described below. The processed vectors are then fed into the RECODER.

The decoder also using the motion vector information from the bitstream compensates each macroblock of each of the frame. The compensated frame is then stored into a frame buffer. Finally each of the frames is stored in the YUV format series file and transmitted to the REENCODER.

2.2 Reencoder's process:

From the YUV files supplied, initially it generates the header information. The frames are passed to the motion_estimation routine in the reencoder. In the **full logic mode** the routine then estimates the motion vectors for each macroblock of each frame. Using these motion_vectors the macroblocks of the different frames are predicted. .If the **frame bypass flag is set** then the transcoder instead of estimation of the vectors retrieves the motion_vector data converted by the CONVERTER and supplies them to the macroblock prediction routine. There are other routines, which do the DCT, quantization and IDCT of various macroblocks. The values computed by these routines is added to each macroblock, and finally the frames are reconstructed.

3 IMPLEMENTATION

3.1 DECODER:

The routine➔Decode_headers extracts the header information from the bitstream and sends it to the ➔Decode_picture which extracts the frame information. The routine Read Bypass.par reads the bypass.par file since the dynamic bypass switch is ON at the beginning. If the Dynamic_Bypass_Flag_Flag_Flag_flag is ON or OFF & DB_counter is 0 then it creates the encoder file. This encoder file has the current values of the bypass.par. These values will be read by the current frame in the encoder. If the FRAME_ bypass _flag is ON then it will create user-defined decoder log called Framedec.txt

3.2 ENCODER:

The control flows to the sequence level routine➔ **putseq**. The user_defined routines reads the enc_Frame_Bypass.par generated by the decoder. The values of the Dynamic_bypass_flag and Frame_Bypass _flag for current frame are read.

If the Frame_Bypass_Flag switch is OFF then it does not create the file (frameready.txt).This file stores the converted data generated by the CONVERTER. The control will be transferred to the routine MOTION_ESTIMATION. The routine PREDICTION will use the motion vector data generated by the MOTION_ESTIMATION routine.

The above sequence of operations can be determined in the Architecture document by following the indicator for the FRAME_BYPASS_FLAG(**yellow box with OFF written on it**) .

Hence similarly following the FRAME_BYPASS_FLAG (**with ON written on it**) you can understand the sequence that will be followed in the program when FRAME_BYPASS_FLAG is ON.

3.3 CONVERTER:

3.3.1 How we converted the original motion vector to encoder's motion vector?

The CONVERTER reads in the motion vector data extracted by the decoder in the file for each frame. Before we describe the mechanism of conversion we need to describe the various attributes associated with a macroblock.

1. Forward motion vectors
2. Backward motion vectors
3. Macroblock_type
4. Motion_type
5. Pict_structure
6. Pict_type
7. Motion_vertical_field_select

3.3.2 Step 1:

The Macroblock_type attribute when extracted by the decoder contains a value. This value represents all the possible information associated with a Macroblock (we will call it Combined value). It includes the motion_vector data values along with some other values combined. The prediction routine in the encoder only needs the motion vector data values associated with Macroblock_type value. Hence CONVERTER extracts only the motion vector data values from the Macroblock_type value. The values of motion_vectors data in the Macroblock_type are:

8 ➔ forward motion vectors if present
4➔ backward motion vectors present
12➔ both the above vectors present(in case of B-picture)

Hence the Macroblock_type value can have any of the above motion vectors values. The motion vectors values are extracted from the Macroblock_type value as follows:

By performing a logical AND of the Macroblock_type with either 8 OR 4 OR 12 where OR is the logical OR.

3.3.3 Step 2:

Macroblock can be decoded as either frame or field type depending on the motion_type associated with it. When a macroblock is decoded as frame, Forward and backward motion vectors of the second field of each macroblock should be zero. That is not done by the decoder for the invalid values .

Hence the CONVERTER takes care of that step by making the second field motion_vector values to zero whenever the MOTION_TYPE attribute has the value 2 .

3.3.4 Step 3:

As we know there is a loss of data because of compression during the original encoding. This results in the invalid values for certain parameters. Decoder is unable to regain the valid values for those parameters since it just extracts what is supplied to it. One of those parameters is the

Macroblock_type associated with each macroblock. The value “0” is considered to be an invalid value for this parameter. It can take any of the following values.

1, 2, 4, 8, 16, 32, or 64. (explained in the files mpeg2dec.h and mpeg2enc.h)

But due to the reason explained above , decoder gets the invalid values for some macroblocks for each frame.

Hence the CONVERTER takes care of the invalid values by putting in the valid values.

4 COMMENTS

In this implementation we decided to use disk files as opposed to socket based inter process communication. The socket based implementation yields faster communication. CCIR-601 4:2:2: 720x480 at 30 frames/second MPEG-2 stream transcoding, with disk file mediated communication, when profiled demonstrated spending 8-10% time in file I/O. The net improvement from socket conversion will impact this part of the overall operation. The focus of this technical report is to document the motion vector bypass mechanism. Please consult associated technical reports for other details.

The work has been funded by DARPA Research Grant F30602-99-1-0515 under it's Active Network initiative.

5 REFERENCES

[1] Bhattacharjee, S., Kenneth L. Calvert and Ellen W. Zegura. An Architecture for Active Networking. High Performance Networking’ 97 ,White Plains, NY, April 1997. [also available at http://www.cc.gatech.edu/projects/canes/papers/anarch.ps.gz, October 98]

[2] Haskell B. G., Atul Puri and Arun Netravali, Digital Video: An Introduction to MPEG-2, Chapman and Hall, NY, 1997.

[3] Information Technology- Generic Coding of Moving Pictures and Associated Audio Information: Video, ISO/IEC International Standard 13818-2, June 1996.

[4] Keesman, Gertjan; Hellinghuizen, Robert; Hoeksema, Fokke; Heideman, Geert, Transcoding of MPEG bitstreams Signal Processing: Image Communication, Volume: 8, Issue: 6, pp. 481-500, September 1996,

[5] Tennenhouse, D. L., J. Smith, D. Sincoskie, D. Wetherall & G. Minden., "A Survey of Active Network Research", IEEE Communications Magazine, Vol. 35, No. 1, Jan 97, pp 80-86

[6] Wetherall, Guttag, Tennenhouse, "ANTS: A Tool kit for Building and Dynamically Deploying Network Protocols", IEEE OPENARCH'98, San Francisco, April 1998. Available at: http://www.tns.lcs.mit.edu/publications/openarch98.html

[7] Youn, J, M.T. Sun, and J. Xin, "Video Transcoder Architectures for Bit Rate Scaling of H.263 Bit Streams," will beappeared to ‘ACM Multimedia 1999’, Nov., 1999. pp243-250.

[8] Fluckiger, F, Understanding Networked Multimedia Applications and Technology, Prentice Hall, UK, 1995.