Published in the Proceedings of
The 3rd IEEE International Symposium on Applications and the Internet, SAINT 2003
Orlando, Florida, January 2003, pp.292-300

Delay and Jitter Minimization in Active Diffusion Computing

Seung S. Yang and Javed I. Khan
Media Communications and Networking Research Laboratory
Department of Computer Science
Kent State University
syang@cs.kent.edu | javed@kent.edu

Abstract

In active computation paradigm the problem of delay and jitter control takes a complex form from its classical counterpart. Here stream processing time becomes a new factor. Also the concept of path bandwidth degenerates as data volume can change while in transit. The paper presents a dynamic feedback based algorithm for this problem. We have recently implemented a concept transport system, where a video stream can receive 'transformation' in a distribute manner over an active subnet while the packets diffuse via multiple paths by a scheme called cooperative transcoding. In this paper, we present the dynamic scheduling algorithm and present its performance on live runs on the test-bed system.

Key words: *content service, active streaming, active network, transcoding, jitter control.*

1. Introduction

The Internet and particularly the Web is increasingly becoming 'active'. Any modern portal already provides significant amount of content adaptation, personalization, and location-aware data insertion [1,2]. The served pages are increasingly combining dynamic information arriving and originating from multiple parties. Indeed, the first generation of a form of *content data networking* (CDN) service has begun with temporary caching proxies of HTTP responses. Now we are seeing increased array of other active services are being added. There is all the reason to believe that there will be an increasing demand for more advanced processing of the incoming information at various intermediate junction points in the network. These points will act as the hub for various actions such as rich domain knowledge based information steering, filtering, multiplexing, adaptation, etc. These will be required as the internet services extends to more diverse platforms (such PDA, wearable computers), devices (such as the emerging Bluetooth devices), and user groups via increasingly diverse infrastructure.

At this beginning stage we seeing the emergence of content services which are HTML oriented. Also now there is no real CDN infrastructure. Therefore the current implementations depend on data processed in an array of backend servers in the content provider's premise. However, several serious investigations are already underway which are looking for more efficient means for performing such active computations within the network. Active Network initiative is looking into smarter router and switch architectures that can support active processing of data packets. IETF Working Group has recently proposed the Open Pluggable Edge Services (OPES) and the Internet Content Adaptation Protocol (iCAP) defining the extended active functions of future caching proxies.

In this backdrop we are investigating issues particular to **active information streaming**. In active information streaming additional design constraints appears pertaining to the temporal characteristics of the information service. Unlike end-to-end streaming, active streaming enables the streamed information to be customized dynamically during its transit. Just like the current HTML based adaptation services, wide range of active services can be potentially built for streamed information ranging from channel multiplexing/demultiplexing, rate adaptation, watermarking, to filtering etc. In this paper we particularly discuss the issue of delay and jitter optimization in Active Information Streaming. As we will show that this problem is quite different from classical networks.

As a test bed for the proposed solution we will use a state-of-the art active streaming system. We have recently implemented a concept prototype—an MPEG-2 **Active Video Streaming** (AVIS) transport [5,7]. From the classical four layers model, AVIS transport is a hybrid network/application transport system. The AVIS transport has been designed to perform arbitrary video transformation over a quasi-active network. The active components follow the flow and can utilize multiple active junctions to perform the required computation. In AVIS transport the video stream receives transformation in a distribute manner in various active nodes over an active subnet while the packets diffuses via multiple paths by cooperative transcoding. Video transcoding is a known computation intensive operations. The type of intermediate processing/service additionally affects the temporal quality of the resulting flow. Because of these challenges AVIS system provides an ideal test-bed for the proposed algorithm.

We first present an overview of related works. In section 3 we then present a brief introduction to the architecture of the AVIS. Section 4 then explains the delay jitter control models, delay estimation process, and the scheduling algorithms. Finally, in section 5 we share performance result of this scheme from live runs of AVIS.

1.1 Related work

The recent schemes proposed for jitter and delay [1,3,4] control can be roughly categorized based on the traffic modeling (statistical vs. observation based adaptation), and the action level (end-to-end vs. network layer techniques). Argiriou and Georgiadis [9] suggested a technique for adapting the transmission rate of an application while maintaining the perceived quality at the receiver at acceptable levels. When a new connection arrives in the system it renegotiates rates for all running applications. Khan, Yang, and Gu [5] proposed the rate symbiosis technique between application and network transport, using an interactive generalization of TCP sending end-point. It does not require any expensive end-to-end measurement. Zhang and Ferrari [4] has presented Rate Control Static Priority (RCSP) scheme, which can provide multi-objective queuing including jitter optimization for Poisson like traffic distribution. In this scheme, a module called regulator is assumed on each stream for traffic shaping based on its optimization objective. A module called scheduler was assumed on switches to resolve the priority for the multiplexed flow. Boorstyn et. al. [1] has presented an algorithm for providing statistical assurance which they call "effective envelop" for traffic scheduling and demonstrated it for optimizing jitter at an intermediate node where multiple video connections between multiple sender and receivers intersect. The method assumes continuous-time fluid-flow traffic. Each intermediate node has one input regulator for each stream, and a scheduler. The regulators and the scheduler work jointly. Bennett et. al. [10] uses special features, called Expedited Forwarding[1] (EF), to guarantee delay jitter bound in Differentiated Services architecture. The packet forwarding at specific rate is guaranteed if (i) a connection is admitted at connection time in EF PHB, and if (ii) the traffic obeys the assumed idealized statistical distribution.

The observation based controls monitor traffics in each node, rather than relying on any static traffic model. The monitored information is fed to a scheduler at switch. Rexford et. al. [3] argues that knowledge of a traffic pattern is not easy to obtain or is limited, like in the case of live video conferencing. They show a Hopping-Window smoothing. Stone and Jeffay [8] described a policy called queue monitoring, which observes delay jitter and dynamically adjusts display latency for low latency conference calls. By monitoring display queue, it retrieves changing end-to-end delay and corrects jitters without time synchronization. Mansour and Patt-Shamir [2] also suggest jitter control algorithms by monitoring buffer fill rates.

Compared to the previous works, this paper addresses the problem of scheduling with respect to joint communication and computation delay. This is unique in active streaming environment. We take the path of dynamic estimation. Active streaming adds a number of new challenges. Even, in a real network environment, it is difficult to obtain the initial traffic model. In active paradigm, network computation adds additional set of complex variability. All network nodes do not have same processing capability. The processing time can vary for different contents and for degree of customization. The initial data can dramatically alter in size and time spacing at each stage of servicing. The capsule data unit can be of unequal size. All packets are not uniformly needed by the service capsules. Also, there is effect of non sequential access. Some of the packets should be used at the same time by the service module, while some others may not be accessed at all. In this paper, we demonstrate a joint buffering and scheduling based algorithm which corrects both computation and transmission difference to reduce the jitter variations to get jitter free play of a video stream.

2. Diffusion transcoder overview

The AVIS system appears as a custom transport between a video server and a set of receiver end-points. The AVIS transport enables two levels of video adaptation. In the first level, based on the link local bandwidth it provides on demand video stream rate adaptation service. However, the video transformation itself is computational intensive. To adapt with respect to the computing power, in the second level, the AVIS transcoder modules can sweep computing power from the neighboring active nodes. For flexible deployment, AVIS have been designed with modular processing components. Where, components can be deployed on separate active

[1] RFC 3246 defines the Expedited Forwarding Per-Hop Behavior (PHB) with the intent to provide a building block for low delay, low jitter and low loss service by ensuring that the EF aggregate is served at a certain configured rate.

nodes. Also, the most computation intensive tasks can be performed dividedly in multiple nodes.

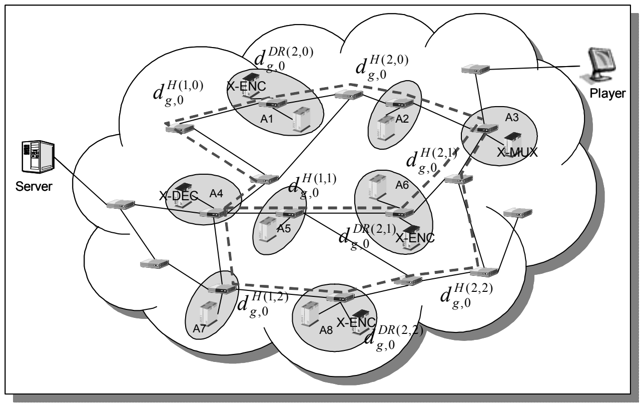

Figure-1 AVIS System deployment model

2.1 Architecture

The principle modules of the AVIS transport has four capsules (i) X-DEC, (ii) X-ENC, (iii) X-MUX, and (iv) X-SPLITTER [5,7]. An X-DEC module decodes an input video stream to decoded frame images. An X-ENC module receives decoded frame images and produces a GOP-ed encoding video stream slice. An AVIS may have several X-ENCs for maintaining proper transcoding rates. An X-MUX is used for guaranteeing the transcoded stream is in sequence. An X-SPLITTER is used for splitting the stream to get a proper branch location for multiple receiver end-point support.

Given the end-points and a network map, with an AVIS transport, the modules X-DEC, X-ENC, and X-MUX are logically connected in a pipe. A particular deployment may have more than one X-ENC between X-DEC, and X-MUX. The X-SPLITTER modules can be located before X-DEC or after X-MUX. Because of the heterogeneous of network environment, some X-ENCs are expected to run on high performance node, while the others on low powered one. The variation in the (i) active processor speed, (ii) the CPU load in the processors from other active processing, and (ii) the difference and variation on the network link/bandwidth in each path from X-DEC to X-MUX via one of X-ENC causes the variance in delay.

2.2 Deployment algorithm

For a given network, and end-points topology the deployment of the AVIS transport components are performed by AVIS Installation Manager (AIM). It uses a mapping algorithm to obtain the initial placement of the capsules in the subnet. The end-application gives initial traffic volume, video compression factor, and the traffic origination and termination points, the end-point constraints such as stream bandwidth in origin/server, and sink/player. Transport designer gives the resource requirements of the capsules such as the required module computation power per mega-bit of video, and modules logical placement sequences. The active junction nodes provide the available overlay bandwidth and computation capacity of the junctions. From these data AIM computes the deployment map. The detail of the initial placement algorithm has been presented in [6]. Here we only illustrate the algorithm with an example deployment.

Table 1. Required bandwidths

End point	Required Bandwidth
Server	30 Mbps
Player	10 Mbps
Original video compression ratio	2 times compressed

Table 2. Connection link status

	Server	A1	A2	A3	A4	A5	A6	A7	A8	Player
Server	-	-	-	-	100 30 70	-	-	100 50 50	-	-
A1	-	-	120 70 50	-	100 30 70	-	-	-	-	-
A2	-	120 70 50	-	100 30 70	100 50 50	-	-	-	-	-
A3	-	-	100 30 70	-	-	100 80 20	300 80 120	-	250 100 100	100 50 50
A4	100 70 30	100 30 70	100 50 50	-	-	200 50 150	-	150 50 100	-	-
A5	-	-	-	100 80 20	200 50 150	-	120 20 100	-	50 30 20	-
A6	-	-	-	300 80 120	-	120 20 100	-	-	300 150 100	-
A7	100 50 50	-	-	-	150 50 100	-	-	-	250 50 200	-
A8	-	-	-	250 100 100	-	50 30 20	300 150 100	250 50 200	-	-
Player	-	-	-	100 50 50	-	-	-	-	-	-

In each cell, the value is for Max BW, Average BW, and Available BW from top to down

Table 3. Active node capacity matrix (r = 1)

	A1	A2	A3	A4	A5	A6	A7	A8
MAX	200	150	150	170	100	300	150	300
Average	80	50	40	20	10	100	60	180
Available	120	100	110	150	90	200	90	120

Table 4. Components' computation requirements

Capsules	Needed Computation
X-DEC	5 per 1MB video stream
X-ENC	17 per output/input stream ratio
X-MUX	23 per X-encoder

Table-1 shows the traffic specification, table-2 shows the network capacity metric, table-3 shows the node capacity metric with radios=1, table-4 shows the AVIS capsule requirements. Finally figure 1 shows a sample deployment.

The tables are used to make a deployment map for the above network diagram. Detail information for the deployment methods is available in [7].

The X-DEC, X-ENC and X-MUX run on active nodes. The number of active X-ENC can vary during run time according to the requirements of transcoding and the capacity of the network [5,6]. Distribution of video stream to X-ENC happens in X-DEC. The transcoding schedule should be available in an X-DEC.

3. Jitter control

3.1 Multipath jitter model

In this section we will first provide a jitter analysis framework for multi-path active computation. This model is generic and is not specific to the AVIS system. In next subsections we will provide the jitter analysis of the AVIS system under this model.

In any active streaming, the delay (and jitter) can happen not only in the network pathway during the transmission, but also in the active nodes during their processing. Therefore overall delay contains computation delays as well as transmission delays. Let g, p, sp, M and H denotes respectively the g-th stream unit, path number, sub-path number, module name, and hop number. Then the delay experienced by an application data unit along a path can be expressed as:

$$Dp = \sum_{i}^{N(h)} \left(d_{g,p}^{M(i,sp)} + d_{g,p}^{H(i,sp)} \right) \tag{1}$$

During the transmission, a stream is processed by a sequence of modules. Each module has a computation delay. This computation delay can be shown as in (2).

$$d_{g,p}^{M(i,sp)} = e_i \times c / s_i \tag{2}$$

Here, e_i is the input stream size in bits, c is a computation needed for the module in flops per input bits. s_i is a speed of the processor running the module in flops. After the stream flows via an active module its size can be potentially modified. The output stream size is represented by a *stage expansion factor*. The stream size after the i-th stage is thus expressed as in (3).

$$f_i = F \times \prod_{j=0}^{i} r_j \tag{3}$$

Let f_i is an output stream size while F is an initial input stream size and r_j is a *stage expansion factor* or output bits per input bits of a module j. The relation of f_i, e_i and r_i is shown in (4)

$$r_i \times e_i = f_i \tag{4}$$

We can get delay in a link as shown in (5).

$$d_{g,p}^{L(i,sp)} = f_i / b_i \tag{5}$$

Here, f_i is an output stream size as seen in (3) while b_i is a bandwidth of link i.

3.2 Multipath timing model

A schedule is divided into computation flow paths. A full flow can have multiple paths. Each network stage in the path involves delay. The figure 2 defines the notation for the arrival times and delays at these stages in a path.

The subscript refers to the application processing unit's (such as GOP) sequence number and a path number, and the superscript refers to the delay stages in i-th module.

- $t_{g,p}^{i,se}$: start time of sending video data time of g-th GOP from server
- $t_{g,p}^{i,ds}$: receiving data and start decoding time of g-th GOP in a decoder
- $t_{g,p}^{i,de}$: decoding end and start sending data to encoder time of g-th GOP in a decoder
- $t_{g,p}^{i,es}$: receiving data and start encoding time of g-th GOP in a encoder
- $t_{g,p}^{i,ee}$: encoding end and start sending data time to MUX time of g-th GOP in a encoder
- $t_{g,p}^{i,ms}$: receiving data time of g-th GOP in a MUX
- $t_{g,p}^{i,me}$: start sending data to player time of g-th GOP in a MUX
- $t_{g,p}^{i,ps}$: receiving data time of g-th GOP in a player

Delays can be calculated based on the times. The encoding stage can have multiple paths. Therefore we will user a second parameter of superscript 'sp' to refer to the particular sub-path in a path p.

- $d_{g,p}^{sd(i,sp)}$: server to decoder transmission delay of g-th GOP; $d_{g,p}^{sd(i,sp)} = t_{g,p}^{i,ds} - t_{g,p}^{i-1,se}$
- $d_{g,p}^{DR(i,sp)}$: decoding delay of g-th GOP; $d_{g,p}^{DR(i,sp)} = t_{g,p}^{i,de} - t_{g,p}^{i,ds}$
- $d_{g,p}^{de(i,sp)}$: decoder to encoder transmission delay of g-th GOP; $d_{g,p}^{de(i,sp)} = t_{g,p}^{i+1,es} - t_{g,p}^{i,de}$
- $d_{g,p}^{EN(i,sp)}$: encoding delay of g-th GOP; $d_{g,p}^{EN(i,sp)} = t_{g,p}^{i,ee} - t_{g,p}^{i,es}$
- $d_{g,p}^{em(i,sp)}$: encoder to mux transmission delay of g-th GOP; $d_{g,p}^{em(i,sp)} = t_{g,p}^{i+1,ms} - t_{g,p}^{i,ee}$

- $d_{g,p}^{MX(i,sp)}$: queuing delay on mux of g-th GOP; $d_{g,p}^{MX(i,sp)} = t_{g,p}^{i,me} - t_{g,p}^{i,ms}$
- $d_{g,p}^{mp(i,sp)}$: mux to player transmission delay of g-th GOP; $d_{g,p}^{mp(i,sp)} = t_{g,p}^{i+1,ps} - t_{g,p}^{i,me}$
- $d_{g,p}^{md(i,sp)}$: mux to decoder transmission delay of g-th GOP; $d_{g,p}^{md(i,sp)} = t_{g,p}^{i+1,ds} - t_{g,p}^{i,me}$
- $d_{g,p}^{sp(i,sp)}$: sub-path delay of g-th GOP; $d_{g,p}^{sp(i,sp)} = d_{g,p}^{de(i,sp)} + d_{g,p}^{EN(i+1,sp)} + d_{g,p}^{em(i+1,sp)}$

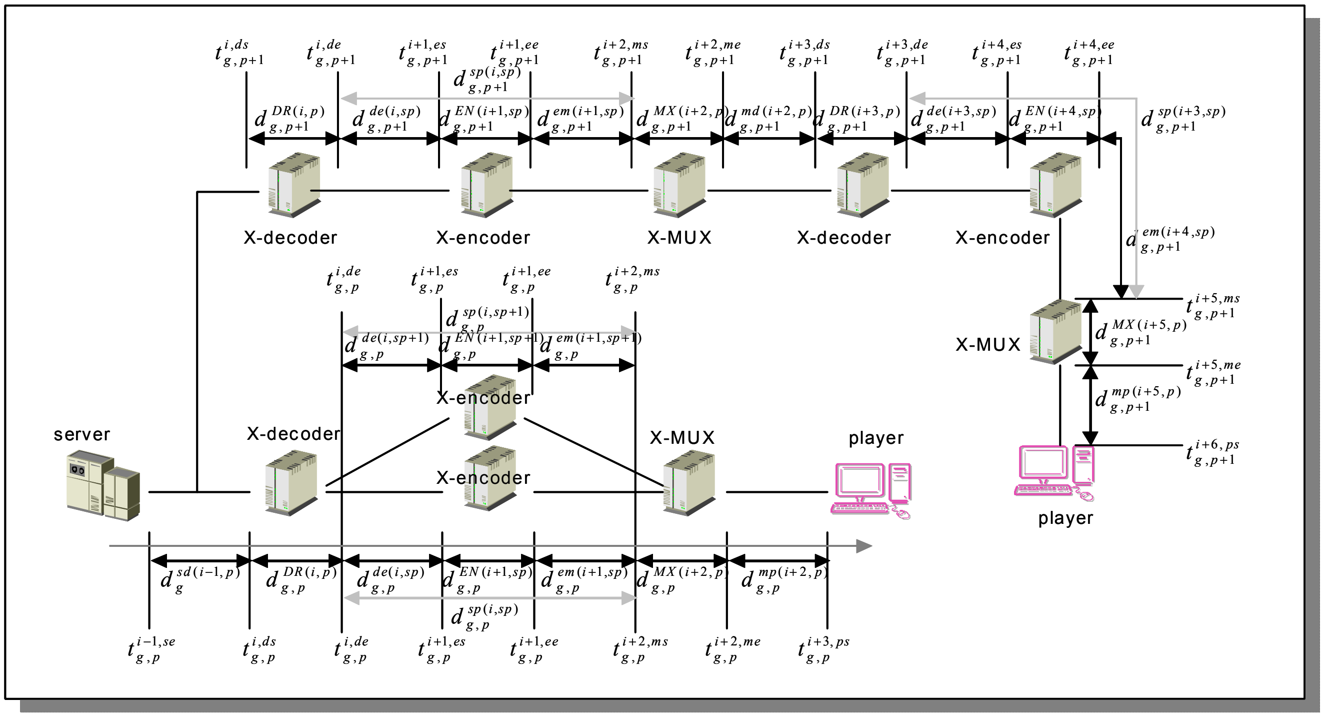

Figure-2 AVIS system time and deploy model

The arrival time of a part of video stream in X-MUX depends on the path it takes. The delay time, $d_{g,p}^{sp(i,sp)}$, represents g-th part of video streams delay which took sub-path sp and is the sum of $d_{g,p}^{de(i,sp)}$, $d_{g,p}^{EN(i+1,sp)}$, and $d_{g,p}^{em(i+1,sp)}$. These are respectively the network transmission delay of decoded video stream, the delay of encoding time, and the network transmission delay of transcoded video stream in sequence.

3.3 Delay determination algorithm

Proper sub-path selection for a given application data unit in a stream is critical to reduce delay jitter in a stream having several processing stages. Selection of proper sub-path is based on the delays measured along the competing paths, i.e. $d_{g,p}^{de(i,sp)}$, $d_{g,p}^{EN(i+1,sp)}$, and $d_{g,p}^{em(i+1,sp)}$. The algorithm chooses a least weighted time path among the paths which has delay less than maximum allowed delay. When there are multiple conforming sub-paths, the weighted time is a time based on *the average time* and *delay variation* of the sub-path. If no sub-path could process a given stream unit (GOP for video) within the deadline (maximum allowed delay for processing this GOP) for it, then the least delay time sub-path is chosen without considering of variations. At the start of the flow, the average delay is initialized to the lowest possible delay of the path and the delay variation is initialized to zero. During the run time, the average time and delay variations are adjusted by measurement on each path taken in the X-MUX. X-MUX gathers delay and delay variation in each sub-path and informs the values to the scheduler in X-DEC. So, even if the initial values are not correct, the algorithm improves the estimates as processing progresses. The following equation is used for deriving expected delays along each sub-path:

$$\tilde{d}_{g,p}^{sp(i,sp)} = \tilde{d}_{g,p}^{de(i,sp)} + \tilde{d}_{g,p}^{EN(i+1,sp)} + \tilde{d}_{g,p}^{em(i+1,sp)} \qquad (6)$$

An expected sub-path delay is the sum of (i) expected delay of transmission from decoder to encoder, (ii) encoding time, and (iii) transmission time from encoder to multiplexer.

As seen in (5), $\tilde{d}_{g,p}^{de(i,sp)}$ and $\tilde{d}_{g,p}^{em(i+1,sp)}$ can be calculated based on f_i and b_i, but f_i and b_i may vary because of several reasons. Such as compression on each application data unit is little bit different from ideal compression ratio or the network activities on a link may cause different b_i values from time to time. So, we use average values based on previous measurements:

$$\tilde{d}^{l(l,sp)}_{g,p} = \tilde{e}_i / \tilde{b}^{l(l,sp)}_{g,p} \tag{7}$$

The bandwidth can be approximated as shown in (8)

$$\tilde{b}^{l(l,sp)}_{g,p} = MAX(b^{l(l,sp)}_{g,p})$$
$$\textit{if no previous GOP was taken the link } l$$
$$\tilde{b}^{l(l,sp)}_{g,p} = \left(\left(\sum_{i=0}^{k-1} b^{l(l,sp)}_{i,p} \right) \middle/ k \right) \tag{8}$$
$$\textit{while } k \textit{ is the number of previous GOP took the link } l$$

Similar to the transmission delay, the module delay can also be different from the ideal expected value. Thus, averages are estimated here as well.

$$\tilde{d}^{m(i,sp)}_{g,p} = MIN(d^{m(i,sp)}_{g,p})$$
$$\textit{if no previous GOP was taken the sub-path } sp$$
$$\tilde{d}^{m(i,sp)}_{g,p} = \left(\left(\sum_{i=0}^{k-1} \frac{d^{m(i,sp)}_{k,p}}{e_i} \right) \middle/ k \right) \times e_g + Q^{m(i,sp)}_{g,p} \tag{9}$$
$$\textit{while } k \textit{ is the number of previous GOP took the sub-path } sp$$

Equation (9) is for delay of a module *m* (*m* is the placeholder for the name of a module, i.e. *d* for decoder, *e* for encoder, and *m* for X-MUX). It is derived from the *average delay per bit* observed on the previous GOP's on the sub-path s*p* and the current input GOP size, e_i. Also, in each module, it has a queuing delay, $Q^{m(i,sp)}_{g,p}$. There is no queuing delay on the decoder (it is relatively fast). An encoder has a queuing delay of (10).

$$Q^{e(i,sp)}_{g,p} = \begin{cases} 0 & or \\ \tilde{d}^{ec(i,sp)}_{g,p} - (T_c - T^{ec(i,sp)}_{s,p}) + \displaystyle\sum_{j=unstarted\ waiting\ jobs} d^{ec(i,sp)}_{g,p} \end{cases}$$
$$d^{ec(i,sp)}_{g,p} : \textit{expected delay of current running job in encoder in sub-path } sp \tag{10}$$
$$T_c = \textit{Current time}$$
$$T^{ec(i,sp)}_{s,p} = \textit{Start time of current job in encoder in sub-path } sp$$

Delay variations of sub-paths are also needed for scheduling of each path. Use of it can provide the worst expected delay time in each path and thus can help in selecting reliable path. Equations (11) and (12) are used to track delay variation.

$$\overline{Avr(d^{m(i,sp)}_{g,p})} = \frac{Avr(d^{m(i,sp)}_{g-1,p}) + \tilde{d}^{m(i,sp)}_{g,p}}{2} \tag{11}$$

$$\overline{Var(d^{m(i,sp)}_{g,p})} = \frac{Var(d^{m(i,sp)}_{g-1,p}) + \left| \overline{Avr(d^{m(i,sp)}_{g,p})} - \tilde{d}^{m(i,sp)}_{g,p} \right|}{2} \tag{12}$$

The delay in the server to the decoder, $d_{g,p}^{sd(i,sp)}$, and the delay in the multiplexer to the player, $d_{g,p}^{mp(i,sp)}$, are simply transmission delays. So, it is retrieved from transmission data size divided by bandwidth of the link. The $d_{g,p}^{DR(i,sp)}$ is the decoding stage delay of the original video stream. In AVIS, the decoding time can be smoothed by changing decoding algorithms. In a practical video coder, however, it is hard to make equal during all video decoding. Decoding time is much smaller compared to the encoding time in transcoding system. Therefore, those transmission and decoding delays are not considered.

The sub-path delay, $d_{g,p}^{sp(i,sp)}$, is used to reduce the jitter at X-MUX. The $d_{g,p}^{sp(i,sp)}$ consists of three delay factors (i) $d_{g,p}^{de(i,sp)}$, (ii) $d_{g,p}^{EN(i+1,sp)}$, and (iii) $d_{g,p}^{em(i+1,sp)}$. These three

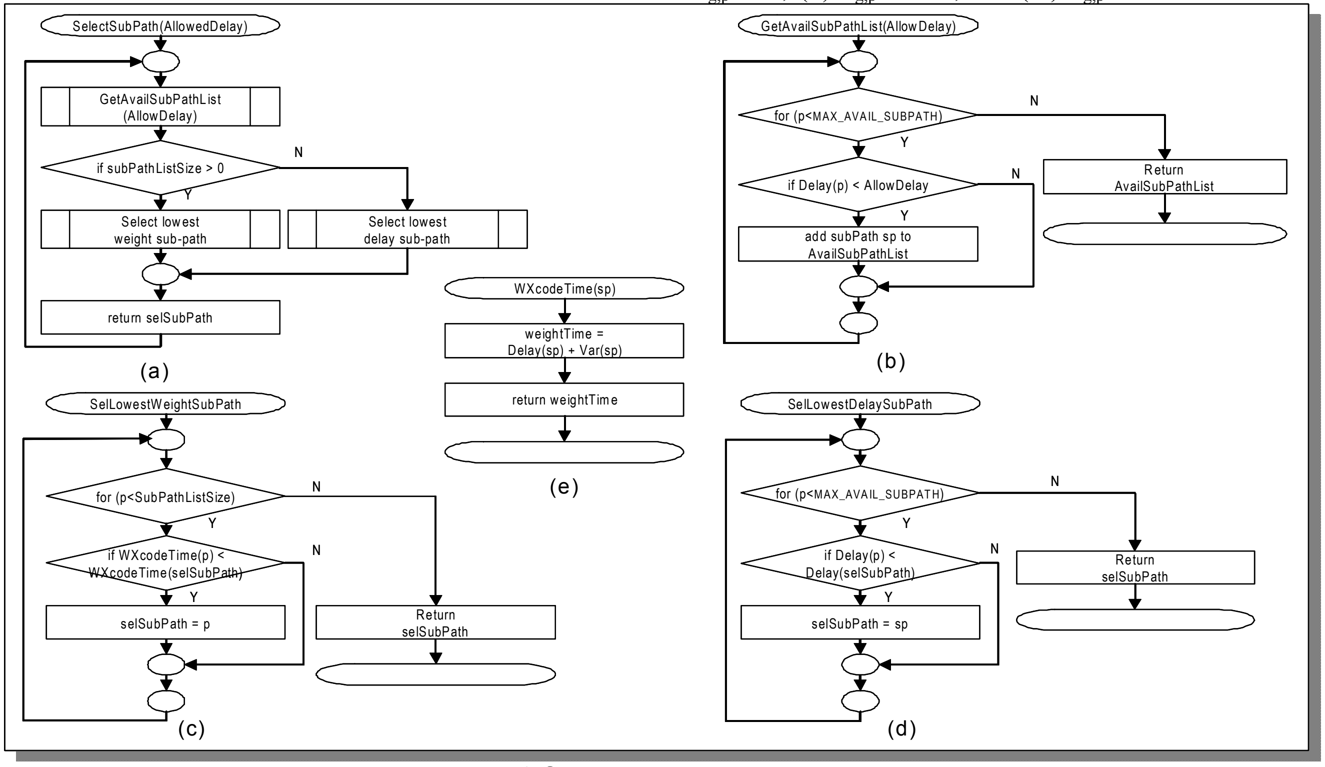
SelectSubPath(AllowedDelay)
GetAvailSubPathList (AllowDelay)
if subPathListSize > 0
N
Y
Select lowest weight sub-path
Select lowest delay sub-path
return selSubPath
(a)
GetAvailSubPathList(AllowDelay)
for (p<MAX_AVAIL_SUBPATH)
N
Y
Return AvailSubPathList
if Delay(p) < AllowDelay
N
Y
add subPath sp to AvailSubPathList
(b)
WXcodeTime(sp)
weightTime = Delay(sp) + Var(sp)
return weightTime
(e)
SelLowestWeightSubPath
for (p<SubPathListSize)
N
Y
if WXcodeTime(p) < WXcodeTime(selSubPath)
N
Y
selSubPath = p
Return selSubPath
(c)
SelLowestDelaySubPath
for (p<MAX_AVAIL_SUBPATH)
N
Y
if Delay(p) < Delay(selSubPath)
N
Y
selSubPath = sp
Return selSubPath
(d)

Figure-3 SubPath selection algorithm

delays vary according to a selected sub-path. A sub-path selection algorithm using $d_{g,p}^{sp(i,sp)}$ will be discussed in scheduling algorithms. The queuing delay in X-MUX, $d_{g,p}^{m(i,sp)}$, do not need to be considered in selecting a transcoding path because the X-MUX. X-MUX absorbs the jitter due to the computation including multi-path flow.

It sends units of stream at fixed rate for a player in normal case. In X-MUX, the buffer size is dynamically estimated to provide the jitter absorption yet keeping the delay minimum. The delay variation and maximum delay is used to get proper buffer size. The following equations are used to get buffer size at X-MUX.

$$\begin{aligned} Max\,(d_{g,p}^{sp(i,sp)}) &= Max\,(d_{g,p}^{de(i,sp)}) \\ &+ Max\,(d_{g,p}^{EN(i+1,sp)}) + Max\,(d_{g,p}^{em(i+1,sp)}) \end{aligned} \tag{13}$$

$$Max(d_{g,p}) = Max(d_{g,p}^{0}, d_{g,p}^{1}, ..., d_{g,p}^{sp}) \tag{14}$$

$$\begin{aligned} MaxVar(d_{g,p}^{sp(i,sp)}) &= MaxVar(d_{g,p}^{de(i,sp)}) \\ &+ MaxVar(d_{g,p}^{EN(i+1,sp)}) + MaxVar(d_{g,p}^{em(i+1,sp)}) \end{aligned} \tag{15}$$

$$\begin{aligned} &MaxVar(d_{g,p}) = \\ &Max(MaxVar(d_{g,p}^{0}), MaxVar(d_{g,p}^{1}), ..., MaxVar(d_{g,p}^{sp})) \end{aligned} \tag{16}$$

$$\begin{aligned} &BufferSize = (Max(d_{g,p}^{sp}) + MaxVar(d_{g,p}^{sp})) \\ &\times No.of\,GOP(per\ \mathrm{sec}) \times AvrageGOPsize \end{aligned} \tag{17}$$

The buffer should have enough data to serve during maximum delay of a transcoding delay and delay variation since it have to serve while it waits a video stream from transcoder.

3.4 Scheduling algorithm

A sub-path is selected to transcode a part of the GOP video stream. The selection is based on the transcoding delay of the sub-path. First, it selects available sub-paths satisfying targeted delay of the GOP video stream. Among the available sub-paths satisfying the minimum delay constraints, it chooses the sub-path with lowest variability. On the other hand, if there is no sub-path satisfying targeted delay, then it chooses the lowest delay sub-path without considering the variability.

3.5 Complexity of the path selection algorithm

The path selection algorithms can be draw like in the figure 3. The time complexity of GetAvailSubPathList(), SelLowestDelaySubPath(), and SelLowestWeightSubPath() take O(sp). The time of WXcodeTime() take O(1). The time of SelectSubPath() = O(sp) + { O(sp) or O(sp) }.

Therefore, the SelectSubPath() takes O(sp) while sp is the number of sub-path.

Generally the number of sub-paths is relatively small. So, the algorithm is reasonably fast.

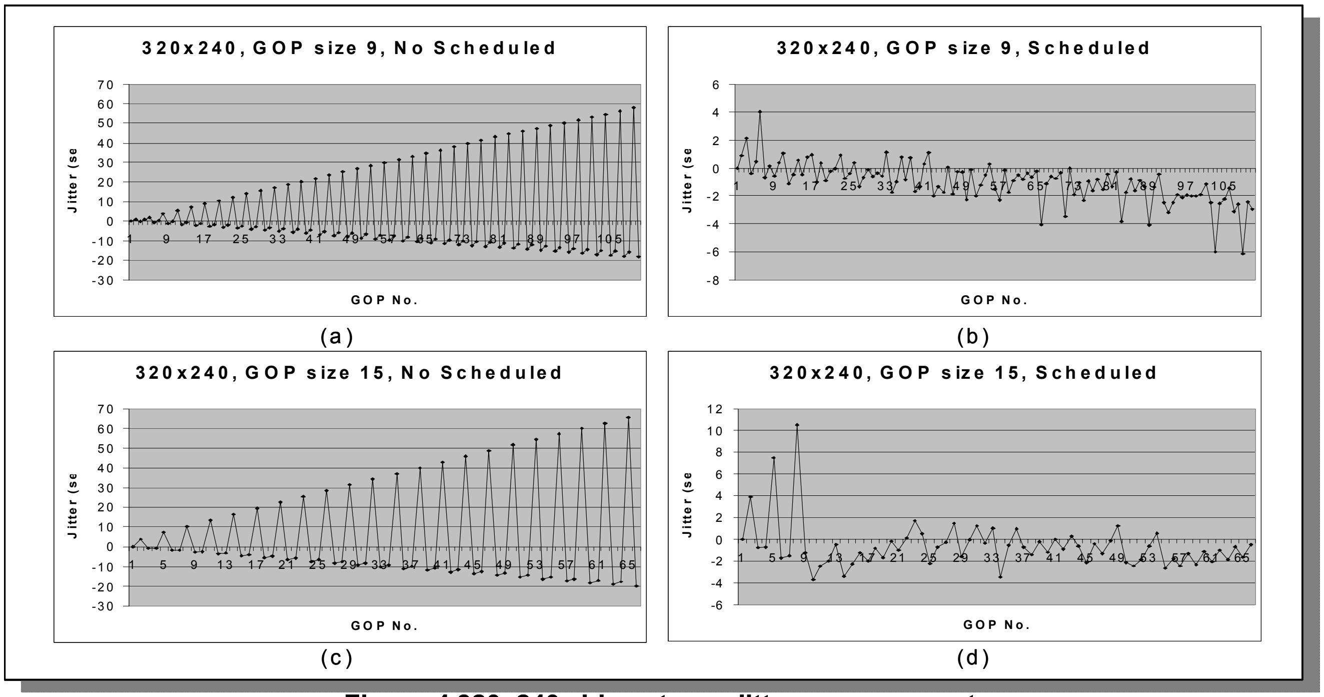

Figure-4 320x240 video stream jitter measurement

Published in the Proceedings of
The 3rd IEEE International Symposium on Applications and the Internet, SAINT 2003
Orlando, Florida, January 2003, pp.292-300

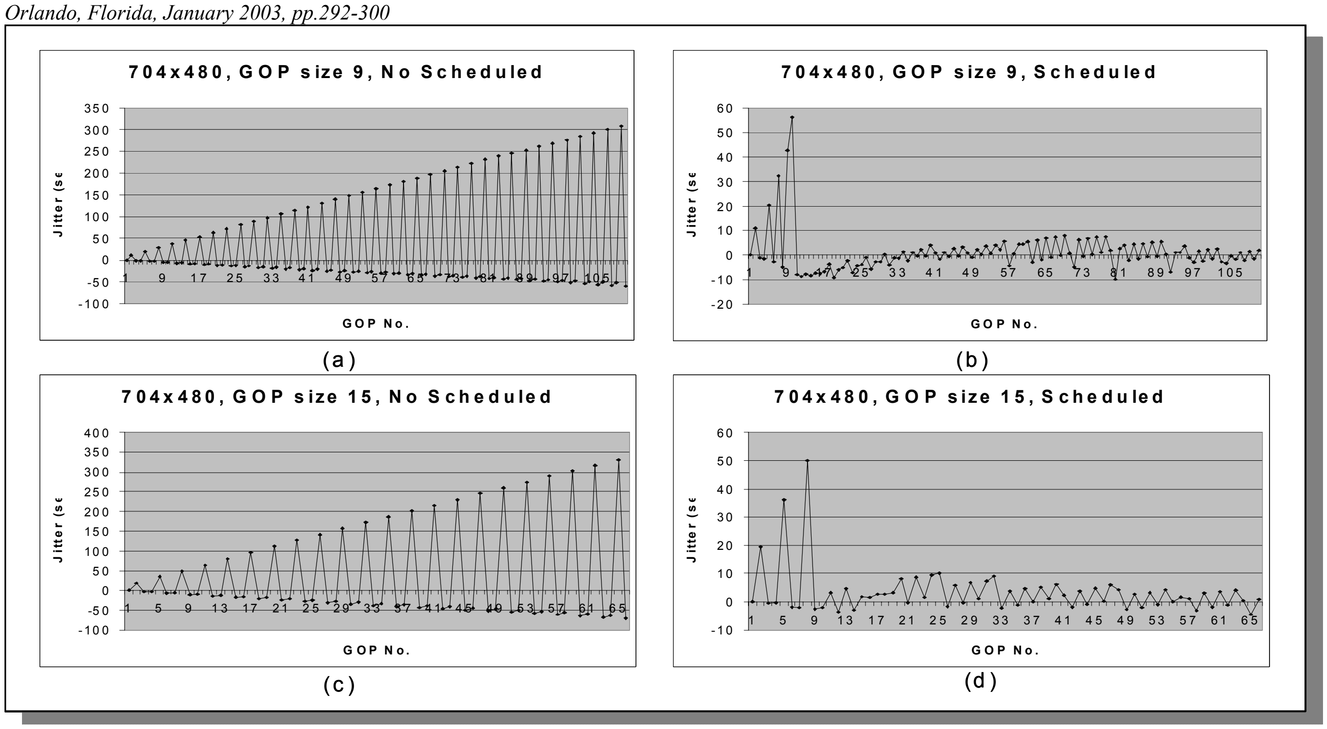

Figure-5 704x480 video stream jitter measurement

4. Experiment results

In test environment, we used total five active network nodes in medianet lab at KSU, each machines running RedHat Linux 7.1. They are three AMD Athlon 1.4GHz, one AMD Athlon XP 1700+, and one dual Pentium III 450Mhz machines. Three encoders are used in the simulation.Those are run on Athlon 1.4GHz, Athlon XP 1700+, and dual Pentium III 450MHz machines. The have different computation powers to make sure that paths have different delay variations in transcoding a video stream. The source video streams have identical contents but different frame and GOP size. There were also other activities on the active nodes because it runs on open active network.

The figures 4 and 5 show the simulation results. The figure 4 plots the jitter performance for both with and without applying the technique. It shows the result of frame size 320x240. The x-axis is the GOP number in the video stream and y-axis are delay jitter in seconds. Figure 5 shows the result of frame size 704x480.

As seen in figures 4 and 5, delay jitters are reduced dramatically with a delay jitter control scheduling. First few GOPs have more delay jitter variations because the scheduler doesn't know proper initial delays of each path. After some time, however, the scheduler adapts in a running environment.

A bigger size of GOP video stream has more delay jitter variations than smaller GOP sized one. This is caused by transcoding method. The encoders start encoding after all needed decoded video data arrived. So, bigger GOP size needs much time waiting in transmission. Also, a bigger GOP stream needs more transcoding time than small GOP video stream. It magnifies the delay jitter variation of delay. So, a bigger GOP sized video stream has larger delay jitter variations. If the encoders can start before all needed video data transferred to it, it will reduce delay jitter more. Also, the results shows that the bigger frame sized video stream has larger delay jitter variations because of the same reasons - much transfer time and transcoding time – and imposed variations caused by contents of a video frame image.

Figure 6 shows the frame-rate observed in their sample run on a small uncontrolled (with background computational and communication load) active network consisting of 5 active routers (with capacity ranging from 400 MHz ~ 1.5 GHz P4 processors, and the interconnections were 10/100 Ethernets with uncontrolled cross traffic). We let the system auto deploy itself and find optimum mapping. Figure 6 plots the frame/ second statistics recorded at the GOP-MUX unit. It plots the performance for both 320x240 and 704x480 frame sizes streams at three different GOP sizes. The computation load heavily depends on the number of macro-blocks or frame size. Based on the frame size the frame transcoding rate varied from 30-5 frames/second.

The adaptive behavior is noticeable at the step like increments at the very beginning. Initially the channel used only one active node. The single node was unable to sustain the target rate. Soon, it auto-deploys additional nodes. For example for 704x480 video the second and the third nodes were deployed some time before 20th and 60th seconds respectively. These delays represent the full feedback and effectuation delays It include (i) the time to detect insufficiency, (ii) the time for stream auto deployment, and (iii) the time it takes the new results to appear at the MUX. As evident from the jumps only three active paths were available. This is dependent on the underlying network configuration.

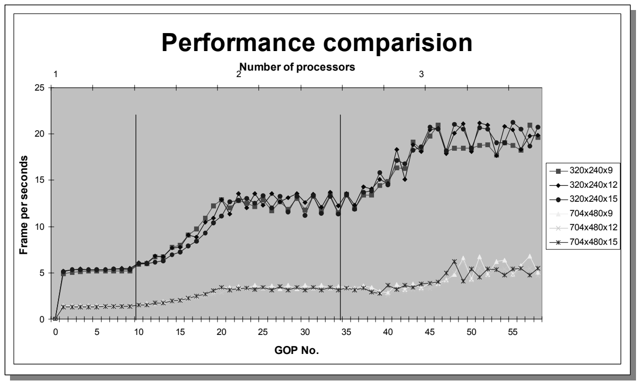

Figure-6 Performance comparison among different frame size and computation

The above results have been obtained from a transcoder running in full motion computation (FMC) mode. While, a general purpose transcoder supporting arbitrary video processing will require full encoding and decoding (used here), more special purpose processing can be performed by domain specific optimized computation. For example, further acceleration is achievable if motion vector computation bypass (MCB) mode is selected. The actual speedup however, is quite complex by the very nature of the paradigm. It will depend on the cost of full motion search which is also configurable, and the ability of computational paths (not only the computing power but also the required bandwidth).

5. Conclusion

Active streaming contains a number of new challenges in network resource management. Computation in the path becomes a major factor in overall performance. Even the notion of bandwidth takes a new form as the volume of data can change as a result of active processing. Almost no previous work exists which considers jitter and delay control in such scenario. With the advent of various content services we believe the problem will be increasingly important.

In this paper, we presented an adaptive scheme which tries to minimize the delay and jitter in this active scenario. In active processing more domain specific optimizations will be needed as opposed to application independent idealized statistical models (such as assumption of Poisson, Fractal, or Normal arrival) used abundantly in classical schemes. Classical closed-box network offers very little support to deploy advanced optimization. An interesting advantage of active streaming is that despite the complexity and application specificity of the probable solutions, the same active technology can be used to deploy such case-specific dynamic and adaptive optimization inside network.

The work has been supported by the DARPA Research Grant F30602-99-1-0515.

6. References

[1] R. Boorstyn, A. Burchard, J. Liebeherr, and C. Oottamakorn. Statistical service assurances for traffic scheduling algorithms. IEEE Journal on Selected Areas in Communications, Special Issue on Internet QoS, 2000.

[2] Y. Mansour and B. Patt-Shamir. Jitter Control in QoS Networks. 39th Annual IEEE Symposium on Foundations of Computer Science, pp. 50-59, October 1998.

[3] J. Rexford, S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, and D. Towsley. Online Smoothing of Live Variable-Bit-Rate Video. In 7th Workshop Network and Op. Systems Support for Digital Audio and Video, pp. 249-257, St. Louis, MO, May 1997.

[4] H. Zhang and D. Ferrari. Rate-Controlled Static-Priority Queueing. In Proceedings of IEEE INFOCOM'93, page 227-236, San Francisco, CA, March 1993.

[5] Javed I. Khan, Seung S. Yang, Qiong Gu, at el. Resource Adaptive Netcentric System: A case Study with SONET – a Self-Organizing Network Embedded Transcoder. In Proceedings of the ACM Multimedia 2001, pp617-620, Ottawa, Canada, October 2001.

[6] Javed I. Khan and Seung S. Yang. Resource Adaptive Nomadic Transcoding on Active Network, International Conference of Applied Informatics, AI2001, Insbruck, Austria, Feburary 19-22, 2001.

[7] Javed I. Khan and Seung. S. Yang, A Framework for Building Complex Netcentric Systems on Active Network, Proceedings of the DARPA Active Networks Conference and Exposition, DANCE 2002, San Jose, CA May 21-24, 2002, IEEE Computer Society Press.

[8] Donald L. Stone and Kevin Jeffay, An Empirical Study of Delay Jitter Management Policies, Multimedia Systems Journal, volume 2, number 6, pp267-279, January 1995.

[9] N. Argiriou and L. Georgiadis, Channel Sharing by Rate-Adaptive Streaming Applications, IEEE INFOCOM'02, New York, June 2002.

[10] Jon C. R. Bennett, Kent Benson, Anna Charny, William F. Courtney, Jean-Yves LeBoudec, Delay Jitter Bounds

Published in the Proceedings of
The 3rd IEEE International Symposium on Applications and the Internet, SAINT 2003
Orlando, Florida, January 2003, pp.292-300

and Packet Scale Rate Guarantee for Expedited Forwarding, IEEE INFOCOM'01, Anchorage, Alaska, April 2001.