Published in the Proceedings of the IEEE/ACM/IFIP International Conference on High Performance Computing, HiPC2003, Hyderabad, India, December 2003, pp.311-322

Delay and Jitter Minimization in High Performance Internet Computing

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

Abstract. Delay and jitter management in high performance active computation is a challenging problem in the emerging area of internet computing. The problem takes a complex new form from its classical counterpart. Here processing time becomes a major part of transit delay dynamics. Also the concept of path 'bandwidth' a classical notion used extensively in classical networking, now degenerates as data volume can change while in transit. The paper presents a dynamic feedback based scheme for this problem with a live internet computing based video transcoder experiment.

1 Introduction

The Internet and particularly the Web is increasingly becoming 'active'. Several paradigms, though emerging from widely different origin, are already underway which are looking for more efficient means for performing active computations with in a network, where the Internet is viewed not only as a large network but also as a confederated platform for integrated computing and communication. The spectrum ranges from grid networking, web-based meta-computing, content services networking, active and programmable networks, sensor computing to the very recent automatic computing [4]. In one end of the spectrum, the Grid initiative is exploring technology so that distributed idle cycles of massive number of computers, including supercomputers, in the Internet can be used to perform advanced scientific tasks [10]. In the context of internet computation, a frequently appearing model of computation is *Active Information Streaming (AIS)*. AIS consider how a passing data stream can be arbitrarily processed while in transit. This data stream does not have to be conventional video or audio stream. Recently, we are investing on various issues related to AIS. Just like the current HTML based adaptation services, wide range of active services can be potentially built for streamed information ranging from channel multiplexing/ de-multiplexing, distributed simulation, remote visualization, automatic rate adaptation, sensor data flow aggregation, to security filtering etc. conforming to AIS model. A particularly interesting problem in AIS is the management of temporal stream characteristics. The problem is quite different and far more challenging from its counterpart in classical networks (we will explain the differences shortly). We believe that it will be a central concern in high performance networked computing irrespective of the framework. This will be a major and central concern for time sensitive application processing, including live simulation, distributed visualization,

live processing of sensor data, instrument control besides media streaming. Interestingly, also many other application processing, which are not normally known to be time sensitive-- may turn into one. The new variability introduced by uncertain compute resources available over a loosely federated resource pool can seriously destabilize synchronization, load balancing, and utilization efficiency of known distributed solutions. In this paper we present a jitter and delay control model for AIS including sharing results from a live experiments on an implemented concept system. We have conducted live video experiments on the recently launched ABONE test bed using a novel video transcoding system called MPEG-2 *Active Video Streaming* (AVIS) transport [5]. AVIS has been designed for arbitrary transformation (or filtering) of a passing video and can be used by any server/player application as a socket like transport. A particularly novel aspect of AVIS is that its processing capable components can utilize multiple processing capable junctions in the pathway to perform the required computation. The video stream can receive arbitrary transformation in a distribute manner in various processing capable nodes over an 'active' subnet while the packets diffuse via multiple paths by cooperative transcoding. Compared to the previous works in jitter and delay control (see [1], [2], [3], [6], [7], [8]) this paper addresses the problem with respect to joint communication and computation delay. No previous work could be found to address this problem. Active streaming adds three new distinct challenges. First, even in a real network environment, it is difficult to obtain the source 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. Secondly, 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. Thirdly, 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.

2 Active Stream Computation Model

2.1 Graph Time Computing

We first present the framework called *Graph Time Computing Model*. A stream transformation occurs in one or more *processing capable subnet* in a path between the source and the sink. The processing of a flow involves an ordered set of transformations on a series of *application data units* (ADU) via a series of sub-task modules. In this *processing capable subnet* the application data units can spread into multiple *processing capable nodes*. The spread occurs to overcome the resource limitation, whether it is the scarcity of available compute cycle on a single node or of bandwidth in an involved pathway. We call the point where the flow enters this subnet as *GT-fork* and the point where it exits the subnet as *GT-joint*. Series nodes are called *GT-connect*. Any AIS layout can be decomposed in terms of these basic connection components. Graph in Fig-1 shows a two level recursive factored graph.

2.2 Flow & Service Model

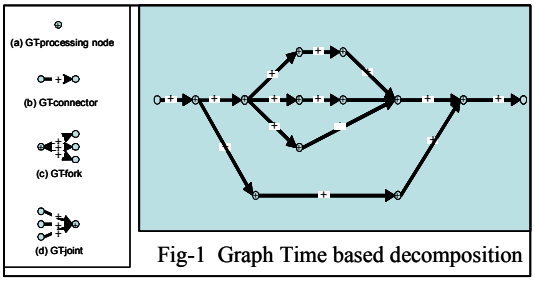

Fig-1 Graph Time based decomposition

In active system, the size and representation of ADU is no longer a fixed quantity rather it can vary between the active hops due to active transformation. Each ADU must pass through the service sub-task modules in same pre-specified order irrespective of the sub-path it takes. We will denote the g-th ADU after the sub-task M as ADU^{M}_{g}. An active stream processing involves a *service*, a *processing capable subnet* platform and a *session* flow. A *service* (A) is characterized by a sequence of sub-task module and their dependency graph <A={a^i}>, with nodes representing the sub-tasks. The *processing capable subnet* is characterized by a graph <N(V, L, C)>, where v^i is the processing capable node, l^{ij} are the overlay links and C is the capacity metric. In C each link has a bandwidth attribute B^{ij} bytes/sec and each node has a compute power attribute B^i flops. A *session* is modeled with a source flow rate <*F*> bytes/second. It is the size of ADUs at the source. Each of the process stages a^i is modeled using its computational and outflow requirements per unit of inflow respectively denoted by r^i_c flops/bps and r^i_d bytes/byte. If b_s bps is the rate at which ADUs are arriving, then $r^i_c.b_s$ flops is the required computational power and $r^i_d.b_s$ bps is the outflow. We assume that the ratios are also applicable of the ADUs.

3 Description of the Test Application

3.1 AVIS Transport

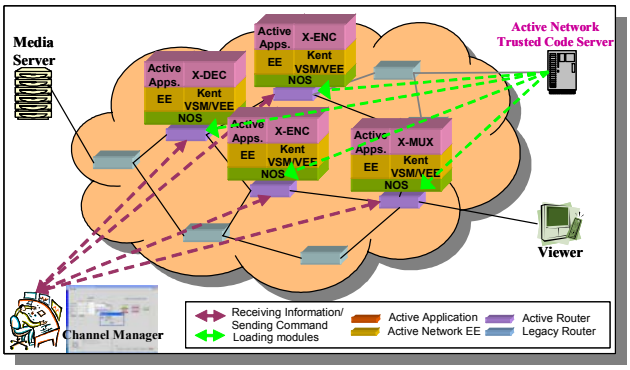

Fig. 2. AVIS system service model

The **Active VIdeo Streaming** (AVIS) system appears as a custom transport between a video server and a set of receiver end-points. It is capable of arbitrary transformation of an MPEG-2 ISO-13818 video stream. The transcoder operates using the compute power of a node (or multiple nodes) in the stream's logical pathway. For this test scenario we demonstrate network embedded rate transformation with adaptive behavior at two levels. In the first level, it adapts video rate based on the available link local bandwidth. In the second level it also adapts with respect to the dynamic variation in available compute power in nodes.

3.2 Architecture of the Test Bed Application

For flexible deployment, AVIS have been designed with modular processing components. Also, the most computation intensive tasks can be performed dividedly in multiple nodes. It has three principle worker capsules (i) X-DEC (GT-fork), (ii) X-ENC (processing node) and (iii) X-MUX (GT-joint) [5]. Besides, it also has a service control capsule called AVIS-Manager. An X-DEC module decodes an input video stream to decoded frame images and schedule to distribute among processing paths. An X-ENC module receives decoded frame images and produces a GOP-ed encoding video stream slice. An X-MUX, is used for guaranteeing the transcoded stream is in sequence. 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. 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.

3.3 AVIS on ABONE

Active VIdeo Streaming (AVIS) system runs on ABONE system using the Kent VSM/VEE execution environment. ABONE is an operational network and provides an Internet wide network of routing as well as processing capable nodes. The Kent VSM/VEE supports object oriented dynamic module management via dynamic loading and unloading of a service composed as a collection of modules, transfer of service configuration scripts, and log files.

4 Jitter Control

4.1 Multipath Jitter Model

First, we will explain the notation. For denoting the delays we use the following two level notations. We use subscripts to refer to the ADU's sequence number (g) and the path number (p). Each ADU is processed by a set of sub-task modules (M). There can be multiple instances of a module. Each sub-path should have a copy of each module. Also, for some services (such as tree-transcoding in a multicast distribution scenario [9]) a stream can encounter multiple services with recurrence of the whole set of transformations. Modules are ordered and have a stage index. Thus, in the superscript each module M is identified with its stage index (i), service number (s) and the sub-path number (sp) within this service. Thus, let g, p, sp, s denotes respectively the g-th ADU, path number, sub-path number, module name, and the

delay stages in a service. Then the delay experienced by an ADU along a path p can be expressed as:

$$D_{g,p} = \sum_{m}^{M} \left(d_{g,p}^{m(i,sp,s)} \right) \tag{1a}$$

For example the transcoding service is defined by the sub-task processing stages $A \subseteq \{F, P, T\}$. Where, F, *P, T* respectively represents the computation delays in the fork, connect and joint units. Their orders are 1,2 and 3 respectively. Thus the total delay stages include the communication delays as well:

$$M \subseteq \{F, fp, P, pt, T\}$$

Here, *fp* and *pt* represent the communication delays in the first and second stages respectively. Thus the objective of the proposed algorithm is to reduce the variation in inter-departure time from the joint defined by (1b):

$$J = \sum_{g}^{Stream} \left| D_{g,p} - D_{g-1,p} \right| \tag{1b}$$

The computation delay of each module can be shown as in (2).

$$d_{g,p}^{a(i,sp,s)} = e^i \times r_c^i / B^i \tag{2}$$

Here, e^i is the input ADU size in bits, r^i_c is a computation needed for the module in flops per input bits. B_i is the processing power on the node allocated to the service in units of flops. The change in size after the stream flows via a processing capable module is represented by a *stage expansion factor*. The size after the i-th stage is thus:

$$f^i = I \times \prod_{j=0}^{i} r^j \tag{3}$$

Let f^i is an output and I is an initial input stream size and r^i is a *stage expansion factor* or output bits per input bits of a stage i. Their relation is shown in (4)

$$r^i \times e^i = f^i \tag{4}$$

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

$$d_{g,p}^{ij(i,sp,s)} = f^i / B^{ij} \tag{5}$$

Here, f^i is an output stream size as seen in (3) while B^{ij} is a bandwidth of link i.

4.2 Algorithm

4.2.1 Scheduling

Given the streaming rate (R) the algorithm estimates a relative *target arrival time* (Tg) at the destination for each ADU. A quantity *maximum allowed delay* is estimated for each ADU based on this deadline. The algorithm chooses a least weighted time

path among the paths which has predicted delay less than maximum allowed delay. When there are multiple conforming sub-paths, the weighted time is a time based on *the average delay time* and the *delay variation* of the sub-path. If no sub-path could process a given ADU within the deadline for it, then the least delay time sub-path is chosen without considering the 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. The *joint* gathers individual delays and delay variations from each sub-path and informs the values to the scheduler in *fork* for subsequent updates.

4.2.2 Delay Estimation

An expected sub-path delay is the sum of (i) expected delay of transmission from fork to the first sub-path processor, (ii) sub-path processing, and (iii) transmission time from last sub-path processor to the joint. The following equation is used for deriving expected delays along each sub-path:

$$\widetilde{d}_{g,p}^{\,sp(i,sp,s)} = \widetilde{d}_{g,p}^{\,fp(i,sp,s)} + \widetilde{d}_{g,p}^{\,P(i+1,sp,s)} + \widetilde{d}_{g,p}^{\,pt(i+1,sp,s)} \tag{6}$$

Links Capacity Estimate: As seen in (5), $\widetilde{d}_{g,p}^{\,fp(i,sp,s)}$ and $\widetilde{d}_{g,p}^{\,pt(i+1,sp,s)}$ can be predicted based on f^i and B^{ij}, but f^i and B^{ij} may vary because of several reasons. The actual compression on each ADU can vary from the ideal compression ratio. The network activities on a link may cause different B^{ij} values from time to time. So, each of the nodes in a sub flow including the joint estimates the average.

$$\widetilde{d}_{g,p}^{\,ij(i,sp,s)} = \widetilde{e}^{\,i} / \widetilde{B}_{g,p}^{ij(i,sp,s)} \tag{7}$$

The bandwidth for each incoming link is approximated by each receiving node, including the fork node, using the method shown in (8).

$$\widetilde{B}_{g,p}^{ij(i,sp,s)} = \frac{1}{k}\left(B_{g(k-2),p}^{ij(i,sp,s)} \times (k-1) + B_{g(k-1),p}^{ij(i,sp,s)}\right) \tag{8}$$

Here k is the number of ADUs which arrived at the receiver node or arrived at the joint using sub-path sp. The g(k) is a k-th ADU number which passed through the sub-path sp. The join estimates the quantity separately for each incoming flow. If the path has no history then last known or initially known bandwidth is used. The right hand side quantities of the equation are observed bandwidth at the joint not prediction.

Processing Capacity Estimate: Similar to the transmission delay, the module delay can also be different from the ideal expected value. Thus, averages is:

$$\widetilde{d}_{g,p}^{\,a(i,sp,s)} = \frac{d_{g(k-2),p}^{\,a(i,sp,s)} \times (k-1) + d_{g(k-1),p}^{\,a(i,sp,s)}}{k} \times e^{i} + Q_{g,p}^{a(i,sp,s)} \tag{9}$$

Equation (9) is for delay of a module *a* (a⊆A). It is derived from the *average delay per bit* observed on the previous ADU's on the sub-path s*p* and the current input ADU size, e^i. Also, in each module, it has a queuing delay, $Q_{g,p}^{a(i,sp,s)}$. In Internet computing all the modules do not operate in identical speed. Each processing module thus

maintains an incoming queue of unprocessed ADUs. For AVIS there is negligible queuing delay on the decoder. It is relatively fast comparing with encoding speed. The encoder's queuing delay is given to equation (10).

$$Q_{g,p}^{P(i,sp,s)} = \tilde{d}_{c,p}^{P(i,sp,s)} - (T_c - T_s^{P(i,sp,s)}) + \sum_{w \in W} \tilde{d}_{w,p}^{P(i,sp,s)}$$

$\tilde{d}_{c,p}^{P(i,sp,s)}$: *expected delay of current encoding ADU, c, in encoder in sub - path sp*

T_c = *Current time* (10)

$T_s^{P(i,sp,s)}$ = *Start time of current encoding ADU, c, in encoder in sub - path sp*

W = unstarted ADU scheduled in the node.

The delay variations of sub-paths are used to select a proper sub-path for a given ADU. Its use provides 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.

$$\overline{Avr}(d_{g,p}^{sp(i,sp,s)}) = \frac{Avr(d_{g(k),p}^{sp(i,sp,s)}) \times k + \tilde{d}_{g,p}^{sp(i,sp,s)}}{k+1} \tag{11}$$

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

5 Experiment Results

In test environment, we used total five ABONE nodes, each machines running RedHat Linux 7.1. The nodes received authenticated transcoding service modules from a code server located at KSU Medianet Lab. Those are run on Athlon 1.4GHz, Athlon XP 1700+, and dual Pentium III 450MHz machines. The deployment, management and monitoring process was automatic and adaptive. Selected nodes have different computation powers to make sure that paths have different delay variations in transcoding a video stream. The selected source video streams have identical contents but were initially encoded with different frame and GOP size. There was dynamic variation on the node and link capacities since there were also other activities on the processing capable ABONE nodes.

5.1 Jitter & Delay Reduction

The fig. 3(a) plots the jitter performance for both with and without 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. Fig. 3(b) shows the result of frame size 704x480. As seen delay jitters are reduced dramatically with the 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. A bigger GOP causes more delay jitter. This is caused by transcoding method. The encoders start only after all needed decoded video data has arrived. So, a bigger GOP size causes larger wait. Also, a bigger GOP needs more transcoding time than smaller GOP. It magnifies the delay jitter variation of delay. If the encoders can start before

all needed video data is transferred to it, it will reduce delay jitter more. Also, the bigger frame stream has larger delay jitter variations because of the same reasons.

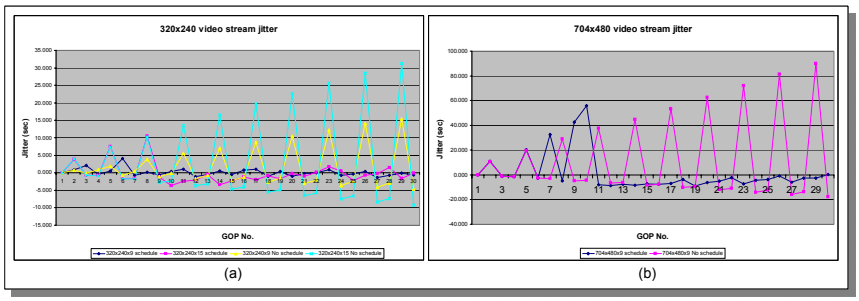

Fig-3 Video Stream Jitter Measurement

5.2 Cold Start Adaptation

Fig. 4 shows the frame-rate observed in their sample run on a small uncontrolled processing capable network, i.e. with background computational and communication load. We let the system auto deploy itself and find optimum mapping.

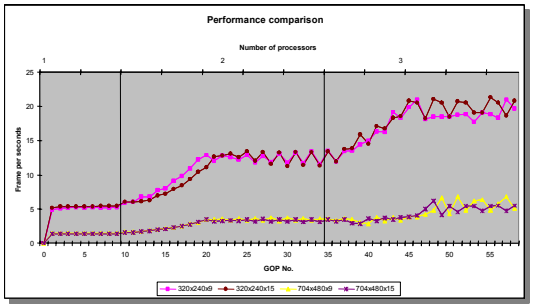

Fig-4 Frame size and computation

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 beginning. Initially the channel used only one processing capable node. The single node was unable to sustain the target rate. Soon, it auto-deploys additional nodes.

5.3 System Deployment and Signaling Overhead

We run the system in three test bed scenarios. In the first scenario, the application as well as the AVIS components-- all were deployed in a single Autonomous System's LAN. In the second scenario the application end points (server and players) were in different networks but AVIS computation was performed in a single network. In the third setup application as well as each AVIS component were in distinct. The corresponding module deployment time of each test bed shows in fig. 5(a). The stack

show how much time was taken by individual components of the system. In all three scenarios the total module deployment took about 1.2 seconds. It includes authentication, automatic module transfer and their activation. The entire synchronization was performed by inter modules signals. Clearly a concern was to how much communication resource was consumed by this. The signaling overhead is plotted in fig. 5(b). It plots individual AVIS components in scenario. For comparison the second bar also shows the actual ADU data volume. Relatively small network resources are used for achieving coordination and controlling among the modules.

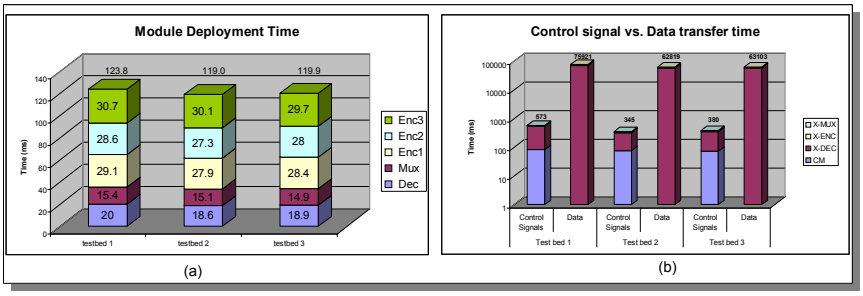

Fig-5 AVIS System Deployment and Signaling Overhead

5.4 Adaptation Performance

Whenever there is a change in the network condition/ capacity the adaptive system responds. The AVIS system offers two forms of adaptation. The first is with the processing capable nodes and their computing powers. Here we provide an experiment on this (as it involves module reallocation) and emulated incremental allocation of additional CPU power in the processing capable nodes into the system in three steps (events T1, T2 and T3) by changing the *target frame rate* of the AVIS system. The corresponding change in X-MUX buffers *throughput frame rate* (FPS) observed in the X-MUX unit is shown in fig. 6. It also shows the reaction time. More computation power gave more performance boost as expected. However, he first effects of the events on the throughput were reflected in about 1.5 to 2.2 seconds. It took little more time before the full effects took place.

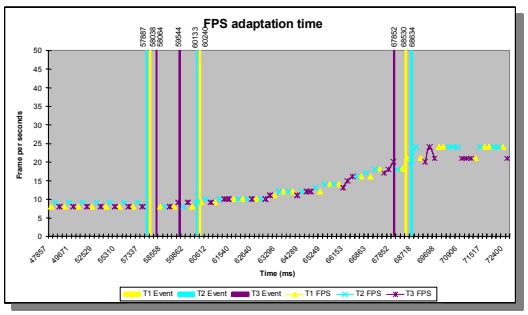

Fig-6 FPS adaptation reaction time

6 Conclusions

It is interesting to note that the prominent target applications in the network computing paradigms (like grid computing, sensor computing or active networking) targets time sensitive systems

such as multimedia, control, or synchronized transactions, where jitter and delay management will be a central issue. The scheme we presented seems to be one of the first to specifically address the issue. However, the temporal QoS required by these applications will be needed in very specific shades and sensitivities. Along with, it is highly unlikely that idealized statistical models (such as assumption of Poisson, Fractal, or Normal arrival), blind to applications', used abundantly in classical attempts, may not be very effective in characterizing AIS where flows are being deliberately manipulated. Therefore, in the design of next generation network for advanced applications it will be of advantage to provision the path scheduling decisions at higher levels- perhaps outside of network layers. The lower network layers should focus more on providing base network parameters to the higher layers to facilitate this decision process. The proposed scheme can be implemented by generic *application level framing* (ALF) with some modification to RTP/ RTCP model. The critical decision that provides the QoS actually lies in the scheduling process. It is possible a different application (such as one which is interested in hard deadline- than jitter) would like to use a different scheduling component in GT-fork. The work has been supported by the DARPA Research Grant F30602-99-1-0515.

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. 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.
3. H. Zhang and D. Ferrari. Rate-Controlled Static-Priority Queueing. In Proceedings of IEEE INFOCOM'93, page 227-236, San Francisco, CA, March 1993.
4. Jeffrey O. Kephart & David Chess, The Vision of Autonomic Computing, IEEE Computers, January 2003, pp.41-50.
5. Javed I. Khan and Seung. S. Yang, A Framework for Building Complex Netcentric Systems on Active Network, Procs of the DARPA Active Networks Conference and Exposition, DANCE 2002, San Jose, CA May 21-24, 2002, IEEE Computer Society Press.
6. 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.
7. N. Argiriou and L. Georgiadis, Channel Sharing by Rate-Adaptive Streaming Applications, IEEE INFOCOM'02, New York, June 2002.
8. Jon C. R. Bennett, Kent Benson, Anna Charny, William F. Courtney, Jean-Yves LeBoudec, Delay Jitter Bounds and Packet Scale Rate Guarantee for Expedited Forwarding, IEEE INFOCOM'01, Anchorage, Alaska, April 2001.
9. Seung Yang and Javed I. Khan, Delay and Jitter Minimization in Active Diffusion Computing, IEEE Int. Symp. on Applications and the Internet, SAINT 2003, Orlando, Florida, January 2003.
10. D. Reed, Grids, the Teragrid, and Beyond, IEEE Computers, January 2003, pp.62-68.

Published in the Proceedings of the IEEE/ACM/IFIP International Conference on High Performance Computing, HiPC2003, Hyderabad, India, December 2003, pp.311-322