Performance Analysis of Application-Level Traffic Shaping

in a Real-Time Multimedia Conferencing System on Ethernets

Yeali S. Sun
Department of Information Management
National Taiwan University
(sunny@im.ntu.edu.tw)

Chin-Fu Ku Yu-Chun Pan Chia-Hui Wang Jan-Ming Ho
Institute of Information Science
Academia Sinica
{chinfu, peter, chwang, hoho}@iis.sinica.edu.tw)
Taipei, Taiwan

Abstract

Many real-time, multimedia applications (e.g., teleconfer-encing) have been successful deployed over the Internet and its overlayed multicast backbone network (MBONE). However, poor, unpredictable transfer performance of the continuous media in Internet is common and inevitable due to the intrinsic "best-effort" transport techniques used. Before the full-blown of ATM networks or any other net-works with QoS offerings are in place, Ethernets are still the most popular networks in today's office environments. In this paper, we investigate how application-level traffic shaping (spacing) can help to reduce possibly drastic per-formance degradation of real-time multimedia conferenc-ing applications on a heavily-loaded Ethernet. An experimental testbed is constructed and all the application software is implemented in our laboratory. Performance results measured from the testbed are presented to provide insights of how the performance degradations occur and to illustrate how a real-time application can do to lesson the situation.

1. Introduction

Many real-time, multimedia applications have been suc-cessful deployed over the Internet and its overlayed multi-cast backbone network (MBONE)[1] such as vat[2], ivs[3], nv[4] and vic[5]. However, poor, unpredictable transfer performance of the continuous media in Internet is common and unavoidable due to the intrinsic "best-effort" transport techniques used. Namely, indiscriminated sharing of net-work resources in Internet makes no guarantee of timely delivery between senders and receivers of real time appli-cations.

End-to-end (application-to-application) timely delivery of continuous media generally requires complex control on the sharing of the communication network resources as well as the resources at end systems according to user-spec-ified Quality-of-Service (QoS) requirements. The manage-ment of network resources includes proper allocation and sharing (scheduling) of transmission bandwidth and switching buffers. The actual performance metrics are such as network transport delay, delay jitter, and information loss rate. Significant research has been carried out in this area for data with real-time characteristics on high speed networks like ATM. Current practice and consensus is that prior reservation of resources is necessary to meet real-time QoS requirements. Various schemes have been proposed (for example see [6][7]) and some of them have been imple-mented in today's ATM-related products. Unfortunately, most of the currently-installed networks do not support timely delivery.

Besides the intermediate networks, proper scheduling of CPU cycles and physical memory to real-time processes (vs. non-real time processes) are as well important to enforce required end-to-end delivery quality. For example, delay jitter of video frames should be properly translated into individual delay requirements at sender's end system, networks and receiver's end system. Namely, a maximum tolerable time ought be specified to govern the processing

433

of digitized audio and video data at the sender from the time instant the data is generated until the time it is successfully transmitted onto the network. Similarly, a maximum tolerable time duration should be specified at the receiver's end system to regulate how fast the corresponding processing threads (a basic unit of CPU control) must process the received data so to achieve timely presentation to the users. Resource control at end systems requires a) kernel support for deadline CPU scheduling[8][9] and the specification of the period and quantum for threads associated with real-time connections (or sessions); b) acceptable protocol processing time at the communication subsystem; and c) tolerable queueing delay at local network interface output buffer. The state-of-the-art operating system design employs microkernel approach and to give priority scheduling to real-time threads, e.g., POSIX p-threads[10].

Several transport protocols have recently been proposed for real-time information delivery such as RTP[11], ST-2[12] and Tenet[13]. Although these protocols have continuously been revised based on the actual operational experiences, two schools of approaches are generally taken in the support of real-time multimedia applications. One approach presumes the underlying networks can be of different types - some may support guaranteed timely information delivery and some may not. As a result, they consider that multimedia applications themselves must equip necessary protocols to manage the real-time data delivery. The other approach is promoted by the people in communications society who consider guaranteed certain degree of QoS delivery is best handled by the transport networks, e.g., ATM, so to minimize the complexity of the applications. In reality, many different networking technologies coexist in Internet and most of them do not provide QoS service. Therefore, most of the currently proposed real-time transport protocols consist of two parts of protocols: a data transfer protocol which defines the format and necessary control information for media data delivery and a control protocol to manage the transfer quality of the media. Nevertheless, each of these proposed real-time protocols has its own design philosophy, thus strength and weakness in supporting real-time multimedia applications across an internetwork. For example, RTP exercises no control over the transfer quality. Instead, it provides mechanisms, i.e. sender report and receiver report, to convey statistical information about the data delivery between senders and receivers and let both parties to decide control policy and have the freedom to exercise any control based on the collected data. The advantage is that RTP makes no assumption about the services provided by the underlying transport networks, completely reflecting the current situation in Internet. If the underlying subnetwork is Ethernet, the guarantee of QoS would become application's respon-

sibility; RTP then provide sufficient mechanisms to allow applications to take necessary QoS handling procedures. On the other hand, when the underlying network is ATM, itself provides service classes to guarantee delivery performance, applications can be relieved largely with how to guarantee network delivery quality. For most of the existing networks like Ethernet, the approach taken in Tenet takes completely different approach. A couple of complicated protocols and mechanism are devised to control bandwidth allocation for the real-time transfer requests and provide rate control, jitter control and transmission scheduling based on QoS parameters. An issue raised here is whether it is necessary to have similar functions (e.g., QoS control) repeated at the different layers possibly resulting in performance overhead and implementation complexity. Still, a number of issues on the design of a truly appropriate real time protocols need to be resolved before the full potential of these new applications can be realized.

Before the full-blown of ATM networks or any other networks with QoS offerings are in place, Ethernets are still the most popular networks in today's office environments. Multimedia conferencing systems are emerging as a tool for effective meeting. It is important and necessary for us to evaluate and understand the performance of running multimedia conferencing applications over Ethernets. Because of the contention-based medium access control scheme (CSMA/CD) used on Ethernet, in general, when network load is high, poor transmission performance is inevitable. In fact, at the initial setup of the testbed, we observed severe video packet dropping under heavy loads and poor audio reception quality competing network resources with video packets.

In this paper, we show how traffic shaping (spacing) can be used in multimedia conferencing applications to reduce performance degradation under heavy network loads, *while* maintaining acceptable transmission and reception quality of the audio and video information. An experimental testbed is constructed and all the applications software is implemented in our laboratory. Performance results measured from the testbed are presented to provide insights of how performance degradations occur and to illustrate how a real-time application can avoid the occurrence of such situations.

This paper is organized as follows. In Section 2, we describe in detail the experimental testbed including the hardware configuration and the structure of the multimedia conference software implemented. In Section 3, an application-level traffic shaping scheme is presented. A queueing model is presented to illustrate the factors that affect the overall (end-to-end) delivery performance. Three

major factors were identified and extensively tested in the testbed. The first control strategy is to perform traffic shaping at traffic source to avoid local buffer overflow due to the bustiness of the generated video traffic. The second strategy is to implement a real-time multimedia application by a multi-threaded process so that user can give higher priority scheduling to real-time threads. This is because we consider the receiving audio information is more important than receiving video images without audio in teleconferencing applications. In Section 4, we present the performance results of our experiments. Finally, Section 5 gives the conclusion.

2. Testbed

The advancement of modern hardware and software technologies has made real-time, multimedia applications and services over conventional computer networks such as Ethernets and Internet possible. One such application is the use of conferencing applications at office environments for effective meeting. In view of these, we built an Internet-based multimedia testbed called *Academia Sinica multimedia Interactive System (ASIS)* to study problems and issues relevant to the design of communication systems, including network protocols, and operating system support for real-time applications. The first attempt is a prototype multimedia conferencing application. This system is implemented on inexpensive equipment listed in TABLE 1..

TABLE 1. Testbed equipment list

Platform	Product
PC	486 DX2-66 PC
Video device	VPON (JPEG)[a] card
Sound device	VPON (Yamaha chip set)
Network card	3COM 3C503
Operating system	Mach 2.6

a. An FIC-AVCom card contains a C-Cube
   CL550 JPEG image compression processor
   and YAMAHA YMZ263B audio processor

To provide precise timestamps in real-time video and audio packets transmission and reception, we implement a distributed master-slave time protocol to synchronize the clocks of all the stations in the network. The system clock is tuned to have resolution down to 1.111 millisecond. FIGURE 1. shows the configuration of the testbed.

The multimedia conference software implemented in our

testbed consists of two parts: a real-time data transfer application and a conference session control application. FIGURE 2. shows the protocol structure of the prototyped system. The UDP protocol is used for real-time data transmission and the TCP protocol is used for reliable transfer of conference session control messages.
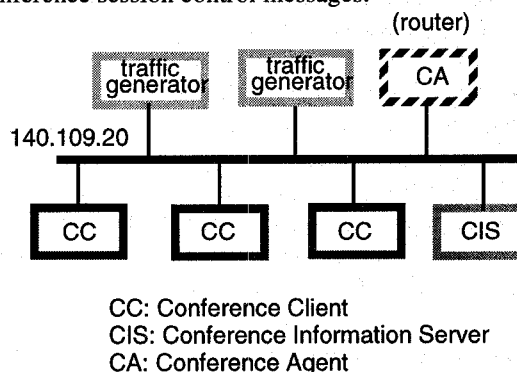

CC: Conference Client
CIS: Conference Information Server
CA: Conference Agent

FIGURE 1. The configuration of the experimental testbed
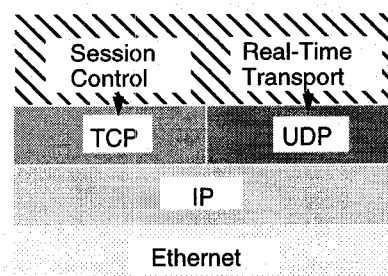

FIGURE 2. The protocol structure of the multimedia conferencing system

2.1 Real-time Data Transfer Protocol

The real-time data transfer protocol provides general packet manipulation functions like media data segmentation and reassembly, as well as error detection (lost, errored and delayed frames). In addition, a traffic shaping mechanism is implemented in real-time application programs to control the sending rate of video packets from the application to network interface output buffer. The objective is to avoid overrunning output buffer by sending large bursts of data to the network interface output queue. Ethernet is a complete resource sharing network with no guarantee of transmission rate to backlogged stations. When the network load is high, a sending station must compete with other stations in gaining network access right. Worse, due to the exponential backoff retransmission algorithm used in CSMA/CD, an active station delays its packet transmission

435

for even longer period of time. If local applications are unaware of the situation and keep sending data to an already heavily backlogged output queue, it will soon be built up, resulting in packet dropping at the sender's network interface. For video transmission, video frames are fetched in a constant rate, e.g., 25-30 frames/second in NTSC and 10 frames/second in multimedia conferencing. Each digitized video frame is usually segmented into a number of packets and send to the network interface as a burst (back-to-back). This procedure may easily overflow the sender's network interface output buffer. To address this problem, we propose to let applications, particularly real-time applications, exercise rate control (traffic shaping) in sending data to local network interface to avoid undesired data loss at the sender's end system.

Data transferred on Ethernet may experience a wide range of delays depending on the traffic load in the network, thus resulting in possibly significant delay variations between consecutive packets generated from a data source. For video transmission, a maximum delay variation tolerance (DVT) is used to indicate the maximum delay jitter allowed between two adjacent video frames in time, denoted by DVT_{video}. Similarly, a maximum delay variation tolerance is set for audio packets, denoted by DVT_{audio}.

The manipulation of video data is as follows: At the sender side, video frames are fetched periodically from video card which is connected to the video camera, e.g., 8 frames/second. Each frame is segmented into a number of 1KBytes blocks. Each block is encapsulated by a header containing a sequence number (a combination of frame number and packet number within the frame), a timestamp and other medium-relevant control information for video playback at receiver. Packets are then ready to be submitted to the kernel for network transmission. The submission rate is controlled by the traffic shaping mechanism implemented in the program. At the receiver, video packets are buffered and reassembled by the receiving application. If any packets of a frame are detected lost or a newly-assembled frame violates DVT_{video} (late frame), the entire frame is discarded and marked as a "frame loss" in the QoS performance statistics (error frames). Blocks of audio samples, e.g., 256 bytes/block are fetched aperiodically. A newly-fetched audio block is sent to the kernel for transmission immediately. Our interface card is ISA which does not have DMA. Therefore each call of "*sendto*" is an I/O operation. The audio thread has to give away its CPU control right. Since Mach 2.6 provides only primitive thread scheduling function, the audio thread will re-gain CPU control only when the other threads explicitly yield the control. A packet header is attached to each sample block

containing a sequence number for reassembly and a timestamp for resynchronization. Note that both audio and video packets have its own timing and sequence number space. Clocks of stations are all synchronized in the testbed. The timing for media to compute jitter is the wallclock time (an absolute time). Audio packets that are detected lost (out of sequence), a "packet loss" is marked in the QoS performance statistics (error audio blocks). If a correctly received audio packet violates DVT_{audio}, the packet is still played out but a "delayed packet" is marked in the QoS performance statistics (late audio blocks).

2.2 Conference Session Control Protocol

A conference session control software is implemented to manage the admission and termination of a conference call as well as the join and leave of participants during a conference. The architecture of the system is shown in FIGURE 3. which consists of four components: Conference Information Server (CIS), Conference Mediator (CM), Conference Agent (CA) and Conference Client (CC). CIS is responsible for the management and coordination of multiple conferences in the system, such as routing plan setup, resource reservation and allocation for each conference. CM acts as a manager and mediator for a conference and handles operations specific to the conference, such as talk-token management and voting management. CA acts as a dispatcher which accepts requests from end-users, i.e. CCs and then forward the requests to either CM or CIS as well as forwards audio/video data to another participant according to a pre-setup routing tables.
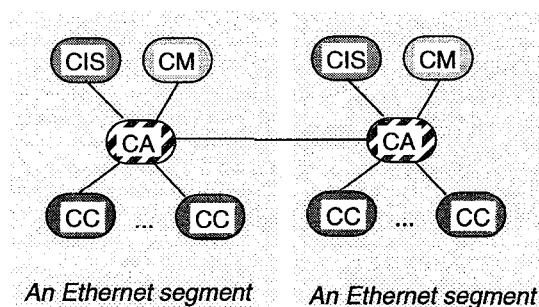

FIGURE 3. The architecture of conference session control system

A CC also provides a graphical user interface through which an end-user can operate the video conference system such as to open, register, join or quit a conference, and to talk or vote. When a user wants to start a conference call, he/she invokes the local CC process to connect to the CA.

He/she then input information about the conference such as the conference name and the start time. These information will be delivered to CA which then forwards it to CIS. Based on the information received from the user, CIS creates a pre-setup routing plan and forks a CM process to handle the conference. Any user who wishes to participate in a conference will first connects himself/herself to CA by invoking a local CC process. He/she then sends a request to CA for a list of currently active conferences. From the list, one can choose which conference he/she wants to join. After joining the conference, CC supports application-level audio/video sending and receiving. The user-side application described in the next section is implemented in CC.

2.3 Multimedia Conferencing Transport Software

The implementation structure of the real-time data transfer protocol and the video and audio information flows are shown in FIGURE 4.. It is implemented as a user-level multi-threaded process to avoid blocking in system calls. There are five threads: *video_send*, *video_receive*, *audio_send/receive*, *command_send* and *command_receive*. The reasons for having separate threads and communication ports for audio and video are 1) to increase the degree of concurrent processing in case that multiple processors are used at end systems; 2) to enable the use of different network paths or network resource allocations, if appropriate; 3) to enable the monitor of the QoS for different media; and 4) to have the flexibility that a user participating in a conference may receive only one medium if they choose.

The *video_send* thread is implemented as follows: it fetches a video frame from the video card at a constant rate, M frames/sec. Namely, this thread is executed every $1/M$ second. Let us denote this interval by T second. In our implementation, fetching a video frame takes a non-zero time (about 70-100 milliseconds). Frame segmentation and packetization also incur some processing time (less than 10 millisecond). Let these processing overheads be denoted by T_p. To fetch next frame in time, packets of the current frame must be sent to the network interface within T-T_p. This interval is exactly the time buffer (range) that the traffic shaping mechanism can use to manipulate packet submission rate to the network interface. Let this interval be denoted by T_s.

The receiver system is targeted to play out video frames at the same rate as the sampling rate at the sender system, i.e. $DVT_{video} = T$ second. The *video_receive* thread compares interarrival time of two consecutively received frames with

DVT_{video}. A delayed frame is marked if

$$(R_i - R_{i-1}) > DVT_{video}$$

where R_i is the arrival time of ith frame at the receiver. Delayed frames result in visible unsmooth, broken scenes. The violation is caused mainly by the network delay.
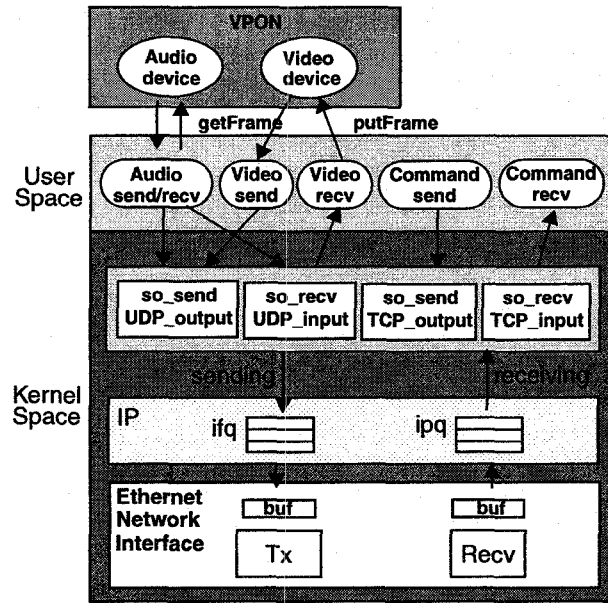

FIGURE 4. The software structure of the real-time multimedia conference application

Operating system support of user-level thread scheduling is important to real-time applications to enforce required end-to-end delivery quality. Mach 2.6 provides only primitive thread scheduling function. A Mach 2.6 thread will continuously hold CPU control unless it explicitly calls the *cthread_yield()* system call. In this environment, controlling the period and quantum needed for real-time threads becomes difficult and complex. For multimedia conferencing systems, it is considered that audio data should have higher transmission priority than the video one when network is congested and the delivery QoS has to be scarified. In our implementation, video threads always call *cthread_yield()* to yield CPU to the audio thread when a video frame has been sent to the kernel or a received frame has been played out. The audio thread only gives up execution right when it finishes processing of a sample block. This, in general, gives the audio thread higher priority than video threads. However, according to our experiments, the

choice of the block size of audio samples on each fetch may result quite different reception quality. For example, system that uses block size of 256 bytes generate good audio reception quality under almost all range of network loads but stem that uses 1024-byte audio blocks produces broken words at receivers.

3. Application-Level Traffic Shaping

In Ethernets, because of the contention-based medium access control scheme - CSMA/CD, when network load increases, data transmission and reception performance degrades. In our case, real-time video and audio packets, together with other non-real time data packets may be simultaneously queued at the local interface output buffer, waiting to siege the channel access right. Because of the limited capacity of the network output buffer, if application processes are unaware of network transmission status and keep sending packets to the output buffer, it will easily overload the queue, resulting in severe packet dropping (loss) at the sender's network interface. In practice, almost all the conventional operating systems like UNIX and Mach do not provide such information like the queue length of the network interface buffer to the applications (i.e. no relevant system calls exist). Moreover, from our experience of packet switched networks in the past, the rule of thumb to avoid congestion at a buffer queue is not to send a big burst of data at one time, especially when network is heavily loaded. In our case, because a video frame is usually pretty large and needs to be segmented into a number of packets suitable for network transmission, we think possible output buffer overrun can be pre-cautioned by equipping some flow control mechanisms in real-time applications. Thus, they would take self-constrained actions in traffic generation into the network interface as well as the network. In this paper, we propose an application-level traffic shaping (spacing) mechanism to control the sending rate of the video packets into the kernel. Instead of sending a bulk of packets belonging to the same video frame, they are divided into a number of smaller groups and spaced out their submission times to the kernel. The goal is to reduce the degree of burstiness of the traffic arriving into the local system's interface output buffer. This is especially important when network load is high, i.e. interface output rate is low and the buffer queue starts to pile up. Through extensive experiments on our testbed, results show that proper setting of the burst size and spacing interval of real-time media like video is greatly helpful in avoiding drastic performance degradation, i.e. delay jitter and packet loss, in Ethernets under heavy traffic loads.

Two parameters are defined in the traffic shaping method:

size of a burst, denoted by S_{burst} which is to manages the size of packets sent in each sending event; and the *constant inter-burst interval*, denoted by T_{burst} which is to control the packet sending rate from the application into the buffer so to avoid overflowing the Ethernet output buffer. The resulting video traffic arriving into the interface output buffer queue can be modeled by a tandem queueing network as shown in FIGURE 5.. Node 1 models the traffic shaping mechanism in the application-level transport protocol software of the video thread. Let γ_{vf} be the video frame generation rate (frames/second) and h be the average number of packets per video frame(packets/frame). Note that the number of packets generated from a frame is dependent upon the contents of the frame. Node 1 is a $D_{bulk}/D/1$ queueing system[14]. The arrival process has bulk arrivals with constant interarrival time T where $T = 1/\gamma_{vf}$. The server in Node 1 models the traffic shaping task by taking multiple number of packets into service at a time. This shapes the arriving burst of average number of packets h into groups of packets of size S_{burst}. Thus, a frame will produce h/S_{burst} bulks. The service time at the server is a constant and represents the interarrival time of bulks that belong to the same frame, i.e.

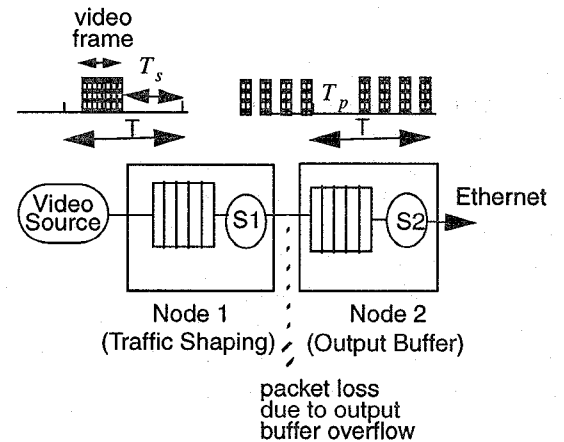$$T_{burst} = \frac{T_s \cdot S_{burst}}{h}$$

FIGURE 5. The queueing model of video packet transmission at the sender's end

Node 2 models sender's network interface output buffer queue. In reality, there are also audio packets as well as other data traffic generated by the local application processes. Since the goal of the analysis here is to evaluate the

438

effect of using traffic shaping on the video transmission performance, for simplicity and mathematical tractableness, we assume only video packets are generated and queued at local output buffer. Thus, S2 can be modeled as a $G_{bulk}/G/1/K$. The arrival process has two states: active and idle. At the active state, packets arrive output buffer in bulks at constant interarrival time. The duration of the active state is a constant of T_s. The duration of the idle state is also a constant of T_p. The service time at Node 2 is very complex. It is the time for a packet to be successfully transmitted over Ethernet when it becomes the head-of-line (HOL) of the output buffer queue. It is a time function of several dynamic factors such as the number of busy stations in Ethernet, the total traffic load in the network, and how Ethernet frames from this station collide with those of the other busy stations. The exponential backoff algorithm will determine how long a collided packet should wait until it is tried again. Also note that the buffer capacity is limited. Therefore, there is a non-zero probability that packets may be dropped due to output buffer full. The queueing analysis of this system is of great complexity. In the next section, performance results measured from the testbed is presented.

4. Performance Results

A series of experiments are conducted on our testbed to investigate the performance of using the proposed traffic shaping method on video data transmission and priority scheduling of network transmission for audio and video data. In the following we show some representative results.

A three-party multimedia conference is established on our Ethernet-based testbed network. The network load is the sum of background UDP traffic plus the traffic generated by the three stations in the conference. Video frames are fetched from video card every $250ms$; a video frame is segmented into an average of 6 packets depending on the movement of persons in the conference; each video packet is of 1K bytes long; and T_p is about 100ms. Three traffic sending strategies, two of them employing traffic shaping, are studied and compared in the following experiments. FIGURE 6. shows the packet transmission patterns of the four strategies. In Strategy A, packets of a video frame are equally spaced out within T_s, i.e. $S_{burst}^{(A)} = 1$ and $T_{burst}^{(A)} = 25ms$. In Strategy B, packets of a video frame are clustered into groups of size 3 whose sending times are equally spaced within T_s, i.e. $S_{burst}^{(B)} = 3$ and $T_{burst}^{(B)} = 60ms$ (it is 60ms rather 75ms because we found the processing overhead T_s increases to about 1140ms. In Strategy C, there is no traffic shaping; all the video packets of a frame are sent to the kernel back to back, i.e. $S_{burst}^{(C)} = 6$. Besides the conferencing application, a UDP

packets traffic generator running on two other stations are used to produce the "*background*" traffic over the network. In the literature, studies have showed that in a typical TCP/IP-based network, 85% traffic is TCP. We know the TCP protocol employs slow start and congestion avoidance window flow control schemes which will throttle TCP packet sending on a congested (or heavily-loaded) network. In the testbed, we want to study the worse case performance of transmitting real-time data on Ethernets, therefore, we choose UDP protocol for the generated traffic rather than TCP. The effect of the proposed traffic shaping method is expected to perform better than that in actual TCP/IP Ethernets.
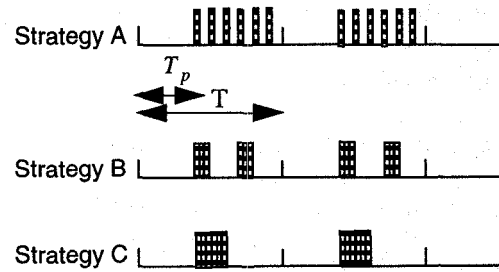

FIGURE 6. Packet transmission patterns of the four traffic sending strategies

FIGURE 7. shows the average video packet loss ratio performance of the three conferencing stations under different traffic shaping strategies. A packet is considered lost if it is never received by the receiver. Such loss is caused by buffer overflow at the sender and/or the receiver's network interface. In Strategy A, packets of a video frame are equally spaced out within the time interval before the next video frame is fetched, namely packets entering the output buffer one by one in a constant rate. The result shows that Strategy A performs well under all range of network load than the performance of strategies B and C, although in B, traffic is divided into two groups and spaced out over 150ms. This is easily understood because under heavy load, the output rate from a station would be slowed down; the output buffer starts to pile up. Suppose the queue is currently full. Under Strategy A, since packets are spaced out, it is more likely that within the packet interarrival time to the output queue that the station will have a successfully transmission. Thus, the next packet arrival to the queue will find a buffer space. In contrast, under Strategy C, since packets arrive the queue in bulk. When network load is high, it is less likely that there will have *sufficient* buffer

439

space left to accommodate a bulk of h packets. Subsequent big bulk packet arrivals would easily overcrowd the queue, resulting in buffer overflow. For Strategy B, its packet loss ratio is better than that of Strategy C when the network is modestly loaded. But its burst size is considered large under heavy loads and therefore has similar packet loss performance with Strategy C. In other words, the inter-burst arrival time, i.e.

In FIGURE 8., we show the frame loss ratio performance which includes frames whose constituent packets were dropped either at the sender's interface buffer or the receiver's buffer due to buffer overflow, or discarded at the receiver due to transmission error. We can see that Strategy A has good frame loss performance at all range of network load. It again shows employing source traffic shaping mechanism can surely improve the quality of the video reception performance.

In this experiment, a total of 5000 video frames are sent which produce about 60000 packets in 21 minutes. During the experiment, we observed quite a number of duplicate packets. Since both Ethernet and UDP/IP do not provide packet sequencing function; the detection of packet duplication and broken video frames (frames with at least one lost packet) as well as frame reassembly are performed by our real-time transport protocol software. TABLE 2. compares the total number of duplicate packets for the three strategies. Strategy A has much less packet duplicates than strategies B and C. Such duplication is mainly caused by the specific implementations of the Ethernet interface cards.

In FIGURE 9. and FIGURE 10., we show the delay performance of video frames under different network loads and traffic shaping strategies. The delay is measured from the instant that the first packet of a video frame is sent to the kernel at the sender's machine until the instance that a video frame is correctly received and assembled by the real-time transport protocol. The average delay of all the strategies remains under the maximum tolerable inter-video frame interval, i.e. 250ms minus frame processing overhead T_p. The average delays for Strategy A remain constant about 150ms. This is understood because all packets of a video frame are sent equally spaced within 150ms. Similarly, Strategy B and C have delays around 75ms and 30ms, respectively. However, strategies with traffic shaping have smaller delay variations than that of Strategy C. But, the value is smaller enough not to affect the end-to-end frame delay requirement. From these two figures, we know that although the average frame delay for Strategy A is higher than the other three strategies but fortunately they are all under the maximum delay tolerance. Most impor-

tantly, it has the least packet loss ratio performance. In FIGURE 11., we show the average packet delay. For the three strategies, the packet delay remains at a rather constant value until the network load reaches 8Mbps. However, we can see the delay for Strategy C is higher than those of the other two strategies. This is because by sending a big burst of packets into the output buffer, the dominant factor of the delay is due to the queueing delay in the buffer. Whereas in the other three cases, packets are spaced out therefore experiencing less delay in the buffer.

5. Conclusions

In this paper, we first described a prototype real-time multimedia conferencing system implemented in our laboratory. It consists of two parts: a real-time data transfer application and a conference session control application. The conference client software is implemented as a multi-threaded process to exploit concurrent processing and to enable the monitoring of the QoS for different media. This design also has the flexibility that a user participating in a conference may receive only one medium if they choose. To implement precise timestamps in real-time video and audio packets transmission and reception, the Mach kernel is modified and a distributed master-slave time protocol is implemented to synchronize the clocks of all the stations in the network. The system clock is tuned to have resolution downto 1.111 millisecond.

By observing severe packet dropping of real-time data in Ethernet and a wide range of delay variations between consecutive real-time packets when the network load is high or when the number of participants or the number of conferences is large. We propose an application-level traffic shaping (*spacing*) mechanism to control the sending rate of the video packets to the kernel. The goal is to reduce the degree of burstiness of the traffic arriving the local end system's interface output buffer, especially when network load is high where the interface output rate is low and the buffer queue starts to pile up. Two parameters are defined in the traffic shaping method: *size of a burst* which controls the size of packets sent in each sending event and the *constant inter-burst interval which* spaces out the sending events into the buffer so to avoid overflowing the Ethernet output buffer.

Through extensive experiments on our testbed, the results show that applications that employ source traffic shaping mechanisms outperform applications that do not in packet loss ratio performance, *while* maintaining acceptable frame delay performance without affecting the reception quality of the video information. The smaller the burst size the bet-

440

ter the result and the best result occurs when packets within a frame are equally spaced out during the inter-frame arrival time (i.e. Strategy A). Such good packet loss performance sustains even under heavy network load.

References

[1]   H. Eriksson, "Mbone: The Multicast Backbone", Communications of ACM, No. 37, Vol 18., pp. 54-60, 1994.

[2]   V. Jacobson and S. McCanne, "Visual Audio Tool", Laurence Berkeley Laboratory. Software on-line: ftp://ftp.ee.lbl.gov/conferencing/vat.

[3]   T.Turletti, "INRIA Video Conferencing System (ivs)",Institut National de Recherche en Informatique en an Automatique. Software on-line: ftp://www.inria.fr/rodeo/ivs.html.

[4]   R. Frederick, "Experiences with Real-time Software Video Compression", 6th International Workshop on Packet Video, 1994.

[5]   S. McCanne and V. Jacobson, "Vic: A Flexible Framework for Packet Video", ACM Multimedia, November, 1995.

[6]   ATM Forum UNI 3.0, "Computer Networks", 2nd edition, 1988, Prentice Hall.

[7]   J. Turner, "Leaky bucket", IEEE Software, Vol. 2, No. 3, pp. 49-61, 1985.

[8]   K. Jeffay, D. L. Stone and F. Donelson Smith, "Kernel Support for Live Digital Audio and Video", 2nd International Workshop on Networks, Operating System Support for Digital AUdio and Video, 1991.

[9]   G. Coulson, A. Campbell, P. Robin, G. Blair, M. Papathomas and D. Shepherd, "The Design of a QoS-Controlled ATM-Based Communications System in Chorus", IEEE Journal of Selected Areas in Communications, September, May, 1995, pp. 686-698.

[10] J. Boykin, D. Kirschen, A. Langerman and S. LoVerso, "Programming under Mach", chapter 10, Addison Wesley, 1993.

[11] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", Internet Engineering Task Force, Audio-Video Transport Working Group, RFC 1889, January, 1996.

[12] L. Delgrossi and L. Berger, editors, "Internet Stream Protocol Version 2 (ST2) Protocol Specification Version ST2+", Internet Engineering Task Force, Audio-Video Transport Working Group, RFC 1819, August, 1995.

[13] A. Banerjea, D. Ferrari, B. Mah, M. Moran, D. Verma, and H. Zhang, "The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences", UC-Berkeley, TR-94-059, November, 1994.
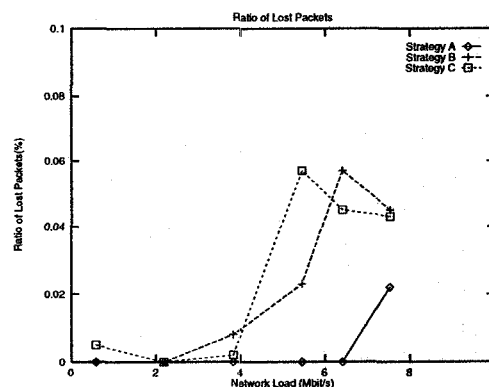
[14] L. Kleinrock, "Queueing Theory:", Volume 1, 1976.

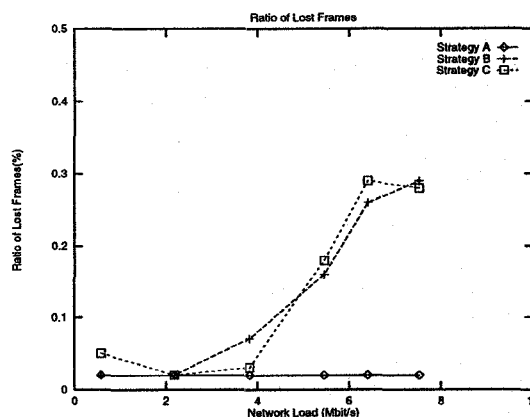**FIGURE 7.**   The video packet loss ratio

FIGURE 8.   The video frame loss ratio

TABLE 2. The video packet duplication ratio

Policy	Backgroud external traffic					
	0Kbytes	200Kbytes	400Kbytes	600Kbytes	800Kbytes	1000Kbytes
Strategy-A	0.00%	0.01%	0.05%	0.04%	0.11%	0.09%
Strategy-B	0.00%	0.20%	0.29%	0.22%	0.15%	0.12%
Strategy-C	0.02%	0.28%	0.32%	0.37%	0.27%	0.24%

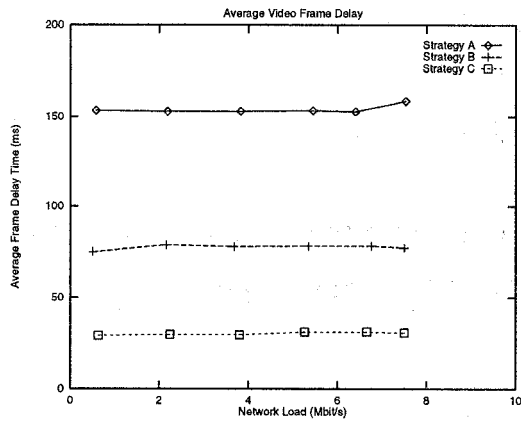**FIGURE 11.** The average packet delay

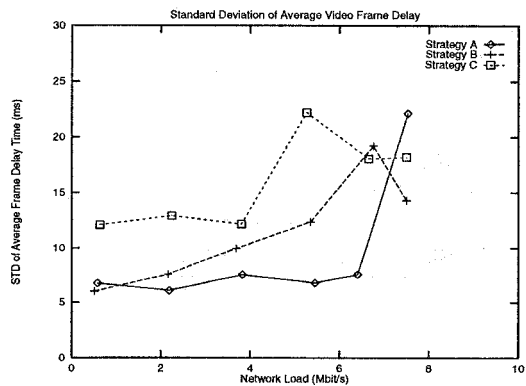**FIGURE 9.** The average end-to-end video frame delay

FIGURE 10. The standard deviation of end-to-end video frame delay