UDC 621.395.74: 621.397.2

Internet Simulator for Testing Networked Multimedia

●Yao-Min Chen

(Manuscript received June 3,1997)

In developing multimedia applications for the Internet, a simulation tool is essential in evaluating the effectiveness of various transmission schemes. We have built such a tool, which uses queueing systems to model network switching nodes. It enables users to control the degree of (simulated) network congestion so that applications can be tested under different traffic conditions. In addition, the tool facilitates subjective testing of continuous signals in real time, so that the perceptual signal quality can be evaluated while network simulation is being run. This feature distinguishes the tool from traditional network simulators.

1. Introduction

Internet has become a popular medium for transmitting continuous multimedia content. However, Internet was originally designed for "best-effort" transmission of data traffic. There are packet loss, packet delay, and delay jitter problems need to be overcome before continuous multimedia applications, such as MPEG audio and video, can be successfully run over the Internet. Fujitsu Laboratories of America (FLA) has been investigating effective schemes to overcome the impact of network packet loss on MPEG audio transmission. The techniques explored include packet-level interleaving, packet loss compensation, and forward error correction (FEC). In evaluating the techniques and their combinations, there is a need for a controlled environment to test their merits. Since a physical testbed is not available, we take the approach of a software *simulator* to emulate the typical Internet behavior. It is a computer program that sits between the end parties of a multimedia communication. It incurs network constraints such as packet loss and delay.

Since subjective perception of media (such as audio and video) is critical in evaluating networked multimedia, a major goal in designing the simulator is to facilitate subjective testing. We believe that the simulator should be designed so that a user can evaluate the quality of the content being transmitted and at the same time observe the network constraints imposed on the transmission. In this way, he or she will more easily capture the correlation between a particular network constraint and the resulting content distortion, and acquire a good sense of how the design can be improved.

As a consequence, our "simulation" is different from the traditional simulation in the sense that simulation events occur in real-time. Some of the events are true events such as the arrivals and departures of *real packets* that contain the multimedia content under evaluation. The others are *virtual events* including arrivals and departures of imaginary packets that contend with the real packets for network resources. An approach related to our idea is the network emulation described in Ref. 1). It used workstations and dedicated Ethernet links to emulate routers and links, respectively, of a wide area network. Since it requires one physical link and workstation to emulate a network hop, it is costlier than our approach because we can aggregate the simulation of several network hops to run on a single workstation. Also, their approach requires modification of operating system kernel, while ours does not.

In our approach, queueing systems[3),4)] are

used to model network elements. A queue (see **Fig. 1**) models a network device that is subject to congestion due to incoming packet flows, such as a network interface at a router or switch. In such a device, typically there is a buffer to accommodate packets arriving too fast over a short period of time, so that the packets are not dropped immediately after congestion. There is also a *server* that services or consumes the packets, such as transmitting the packets over a communication link. Note that if the aggregate arrival rate of the packet flows is persistently larger than the service rate of the server, eventually the incoming packets overflow the buffer and are dropped.

At each queue, a traffic stream (i.e., the data flow from an application) contends with other network traffic for limited resources. The resources include buffer space, communication bandwidth and processing capacity. Due to contention for the resources, the stream suffers packet loss, packet delay and delay jitter, in the following fashion.

Packet Loss:

Since the buffer size is finite, a packet is dropped if the buffer does not have the capacity to accommodate the packet upon its arrival.

Packet Delay:

The delay is caused by the *waiting time* and *service time* of the packet in the queue. The waiting time is the time spent waiting in the buffer before starting being serviced by the server. The service time is the amount of time spent in service.

Delay Jitter:

The waiting time experienced by a packet depends on the amount of data waiting in the queue when the packet arrives. It is a random variable that results in delay variation from one packet to another. The variation is commonly referred to as delay jitter.

When a packet is sent over the Internet, it typically goes through many network devices. Hence, the end-to-end path of the packet can be modeled as a concatenation of queues. Although

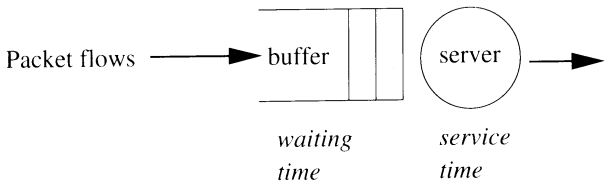

Fig.1– A queue.

the end-to-end delay along the path may be composed of propagation, transmission, queueing and processing delays, it is well known that queueing delay (the waiting times accumulated at the queues) imposes the most critical challenge to applications. This is not only because queueing delay tends to be the dominating component among the four, but also because it is mainly responsible for delay jitter. Delay jitter needs to be carefully handled by applications to avoid buffer underflow or overflow at the playback stations.

It is also well known that buffer overflow at the network devices contributes most significantly to packet loss experienced by applications. Traditionally, packet loss is handled by Transmission Control Protocol (TCP) for reliable transmission. However, for continuous multimedia applications that typically stress timely and regular delivery of data, use of TCP has the following two undesirable effects. First, retransmission of lost packets causes long and widely variated packet delay. Second, the sliding window flow control results in time-varying application throughput. Since these effects cannot be overcome easily, User Datagram Protocol (UDP) becomes a common alternative to TCP. However, UDP-based applications, such as the MPEG audio transmission being researched at FLA, are subject to packet loss caused by buffer overflows in the network.

Therefore, from both the delay and loss perspectives, queueing systems justifiably model the typical network constraints on continuous multimedia applications. This concept is realized by a software implementation in the C programming language over the Sun Solaris operating system. It has been put to use in evaluating the transmission techniques for MPEG audio and shown very

helpful in guiding us to find the right design for improving the quality of the network-transmitted audio.

However, since the simulator relies on the clock timers and interrupt signals of the operating system to generate events, the simulator is subject to the constraint of the timer resolution of the operating system (in the case of Solaris, the resolution is 10 ms). Hence, there is timing inaccuracy. To control the effect of the inaccuracy, a user should not specify simulation parameters that result in events much more frequent than the granularity. It is still an ongoing research issue as how to reduce the effect of the timing inaccuracy on simulation results. Hence, for the time being, the simulator is more suitable for facilitating the design of networked multimedia, rather than being used as an evaluation tool for the design of network infrastructure such as protocols.

The issue of timing inaccuracy could deteriorate when multiple processes contending for the same CPU. Currently we rely on increasing the priority of simulator processes and blocking the access of other users. This is not totally satisfactory. Hence we are looking into adopting the technique of user-level real-time scheduler, such as URsched described in Ref. 2).

Also note that since the project of MPEG audio transmission motivated the development of the simulator, some inter-process messages currently implemented are tied to the MPEG audio project. However, this is only limited to control messages (see Chapter 3). The actual data can be encoded in any format, as long as they are encapsulated into UDP packets. When adopting the simulator to another multimedia application, one only needs to modify the control messages of the application to work with the simulator. Alternatively, he or she can replace interface modules of the simulator with ones suitable for the new application.

We conclude this chapter with an outline of the balance of this paper. Chapter 2 describes how the abstract notion of a queue is implemented in software. In Chapter 3, we describe how multiple queues (that are computer processes) are concatenated using network programming, so as to more truthfully model the network condition along the end-to-end path of a data flow. In Chapter 4, the operation of using the simulator to evaluate the performance of various MPEG audio transmission techniques is presented. Then, Chapter 5 provides concluding remarks and future work.

2. Simulation of a Queue

Figure 2 gives a functional diagram of a UNIX process that implements a queue. There are input and output interface modules that interact with other processes. The input interface receives data packets and exchanges control messages with the sender of the packets. The output interface sends out data packets and exchanges control messages with the recipient of the packets. The main part of the process is the queueing model in the lower portion of the diagram.

For the queueing model, we use two parameters to represent the resource constraints. The first is *buffer capacity*, which represents the amount of storage resource for the queue. The second is *service rate*, which represents the constraint on how fast data leave the system. Buffer capacity models the finite memory size at a network device. Service rate models the finite processing and communication bandwidths for the device.

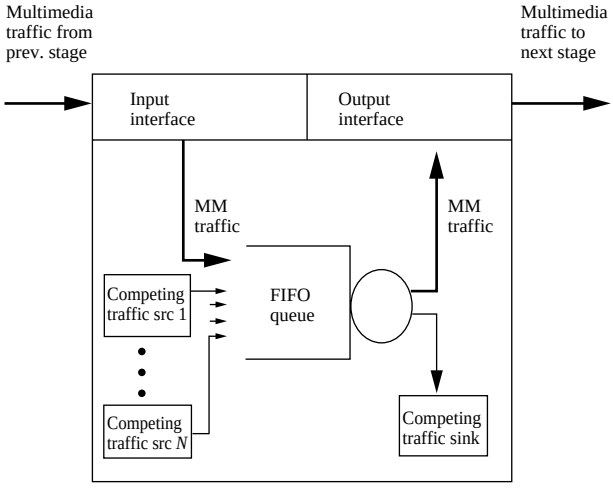

Fig.2– Block diagram of a simulator process.

The system keeps a state called *queue size*. It represents the amount of system storage capacity (such as memory) that are currently *occupied* by data, including the unsent data in the packet currently being served. The data represented by queue size can be either from the real data traffic under evaluation, or from the *virtual* data traffic. The virtual data traffic consists of imaginary data flows that interfere with the real data flow and can cause it to experience queueing delay and packet loss.

Both real and virtual data flows produce packet *arrival events* at the simulation process. The arrival event of a real data packet is triggered when the input interface receives such a packet, while the arrival event of a virtual data packet is generated within the queueing model. When an arrival event occurs, the sum of the packet size p and the current queue size q is checked against the buffer capacity B. If $q + p > B$, meaning buffer capacity is exceeded, the packet is dropped. Otherwise, the queue size is updated by

$$q := q + p. \tag{1}$$

Once the queue size is updated, the departure time t_d for the packet can be computed by

$$t_d := t_a + \frac{q}{C}, \tag{2}$$

where t_a is the event time of the arrival event and C denotes the service rate of the queue. Note that the ratio q/C accounts for the waiting and service times (see Fig. 1) of the packet.

If the packet is real data, the departure time t_d is the *event time* for the *departure event* of the packet. The event time is used to schedule the departure of the packet. When it is due, the packet is forwarded to the output interface.

We proceed to describe the virtual data traffic in more details. The virtual data traffic is modeled as a set of traffic flows called *background streams*. Each stream is characterized by two parameters: *mean packet size* and *arrival rate*. The first parameter describes the lengths of the packets in the stream. More precisely, the lengths are from a random distribution, and the parameter denotes the mean value for the distribution. Currently we have only implemented two distributions. The first distribution assigns to each packet the same length, which we will refer to as the *fixed-sized* distribution. In this case, the length of each packet is equal to the parameter. The other distribution, identified as the *exponential* distribution, uses an exponential random number generator to generate the length of a packet according to the mean packet size. Note that a user can choose, independently for each background stream, whether the packets of the stream have a fixed or exponentially distributed size.

The arrival rate parameter, combined with mean packet size, determines the inter-arrival times between adjacent packet arrivals. Here, we assume that the packet arrival process is Poisson. In other words, the packet inter-arrival times follow an exponential distribution. The mean inter-arrival time τ is computed from the arrival rate R and mean packet size L by

$$\tau = \frac{L}{R}. \tag{3}$$

Whenever a packet arrives, a random number generator is called to generate the inter-arrival time t_i between the packet and its succeeding packet from the same stream. Then, the arrival time t'_a of the succeeding packet is determined by

$$t'_a := t_a + t_i. \tag{4}$$

The arrival time t'_a is the *event time* for the *arrival event* of the succeeding packet.

We can now enumerate the types of events used in the simulator: (*i*) the arrival events of real data packets, (*ii*) the departure events of real data packets, and (*iii*) the arrival events of virtual data packets. The first type is triggered by receiving the real data packets, while the other two are generated internally by the simulation process. A challenge in implementing the simulation process is in the scheduling, sorting, and timely generation of the internal events. Currently we use a linked list to store the set of events waiting to be

generated. The list is referred to as the *event set*. Each event has an event time that is the scheduled occurrence time for the event. If the event is the departure of a real packet, its record also has a pointer to the memory address where the packet is physically stored. The events in the event set maintain sorted in increasing order of their event times. This is done by inserting each new event into the proper place in the event set.

By maintaining a sorted event set, the operating system only needs to use one timer, which keeps track of whether the event time of the top event has expired. The timer is set whenever there is a new top event, which happens after a preceding event expires. The timer is set to be expired in a time value equal to the difference between the current system clock value and the event time of the top event. When the timer expires, the event is *generated*. If the event is the departure of a real packet, the packet is forwarded to the output interface. Otherwise, meaning that the event is the arrival of a virtual packet, the random number generator is called to compute the arrival time of the next virtual packet from the same stream, according to Equation (4). In addition, the queue size is updated according to Equation (1).

Note that Equation (1) assumes that the value of q is kept up to date at every instant. In reality, we need to update the value of q only when a real or virtual packet arrives. When such an event is generated, the following computation is executed:

$$q := \max \{q - C (t - t_0), 0\}, \qquad (5)$$

where t is the current event time and and t_0 is the last event time when q was updated. Note that the updating in Equation (5) needs to be performed before we check whether to drop the arriving packet.

Figure 3 shows the user interface for running the simulation process. In the top left block, the boxes for Buffer Size and Service Rate are where a user enters the resource constraints. The Connection HOST and Connection PORT facilitate communication between processes, which we will describe in the next chapter. The top right block is where the parameters for background streams are entered. For each stream, the user enters the parameters for the stream, pushes the ENTER Stream button, and then switches to another stream. The user can stop this process any time after the parameters for the first stream has been entered. The simulation process automatically keeps track of the number of streams specified. In the example shown in Fig. 3, the user has specified three streams.

The other parts of the user interface incorporate some mechanisms for the convenience of simulation experiments. There is an option of repeating the same sequence of lost real packets as in the previous simulation. In addition, the middle and bottom parts of the user interface dynamically show the packet loss ratio of an ongoing simulation. The packet loss ratio is with respect to the real data packets and is computed for every

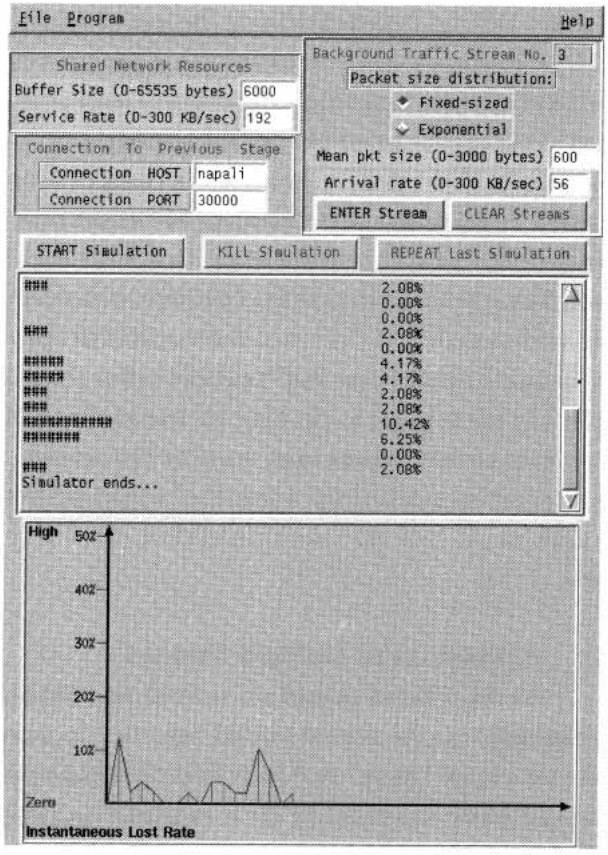

Fig.3– The graphical user interface for simulating a queue.

group of 50 packets. The numerical values for the dynamic packet loss ratio are listed in the middle part of the user interface, and are displayed graphically in the bottom part.

There is also a command mode for entering simulation parameters and invoking a simulation process. More details on both the graphical user interface and the command mode can be found in Ref. 5).

We conclude this chapter with two remarks. First, note that the data structure for the event set belongs to a more general problem called the Event Set Problem[6]. Recently we became aware of more efficient implementation schemes such as the one described in Ref. 7). We are investigating these schemes for future improvement of the simulator.

Second, as we discussed in the introduction chapter, the coarse resolution of the operating system clock timers incurs timing inaccuracy in events. To alleviate the effect of the inaccuracy, we are looking into minimizing the use of timers. In particular, we can restrict the use of timers only to schedule the departure events of real data packets. This can be accomplished by aggregating the events for the virtual data traffic, in the following sense. Let t_1 and t_2 be event times of two consecutive events (arrival or departure) of real data packets. The events for virtual data packets, with event times between t_1 and t_2, are not generated at their scheduled event times but "realized" after the interrupt for t_2 occurs. As long as the realization leads to correct queue size, seen by the arriving or departing packet, the algorithm is correct but relies much less on operating system clock timers.

3. Simulation of Multiple Queues

As we allured to earlier, several simulation processes can be concatenated together to construct a model more truthfully describes the end-to-end path experienced by a multimedia flow. These concurrent processes can be run on the same machine or different machines, which is facilitated by the UNIX Socket programming.

In developing the connection mechanism, we assume a client-server paradigm for the multimedia application under evaluation. Under the paradigm, the application client sends a request to the server to initiate the data transmission. If the server honors the request, it responds by sending acknowledgment and then transmitting the requested multimedia content. When simulation is inserted in the data path between the server and client, we consider the server the most *upstream* process, the client the most downstream, and in the middle a sequence of simulation processes representing a sequence of queues (see **Fig. 4**). Note that the direction of flow is with respect to how actual multimedia data are transmitted. However, along with the data, there are control messages used to coordinate the server and client. These messages can travel upstream or downstream. Specifically, the request message goes upstream, while the acknowledgment message traverses the opposite, downstream direction.

To facilitate the communication between a simulation process and its immediate up- and down-stream processes, it needs to open up a few datagram ports upon starting. It uses a *connection port* (resp., *control port*) to communicate control messages with its immediate downstream (resp., upstream) process. In addition, it uses a *data port* to forward and receive the actual multimedia data. **Figure 5** shows an example of using three queues to simulate the end-to-end path of a networked multimedia application. Note that although we show two data ports within the block of each simulation process, this is just for the sake of presentation clarity. In practice, one data port is sufficient.

A process P, before sending any message to its immediate upstream process, needs to know

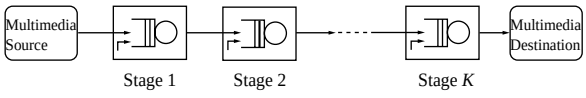

Fig.4– The concatenation of multiple simulation process.

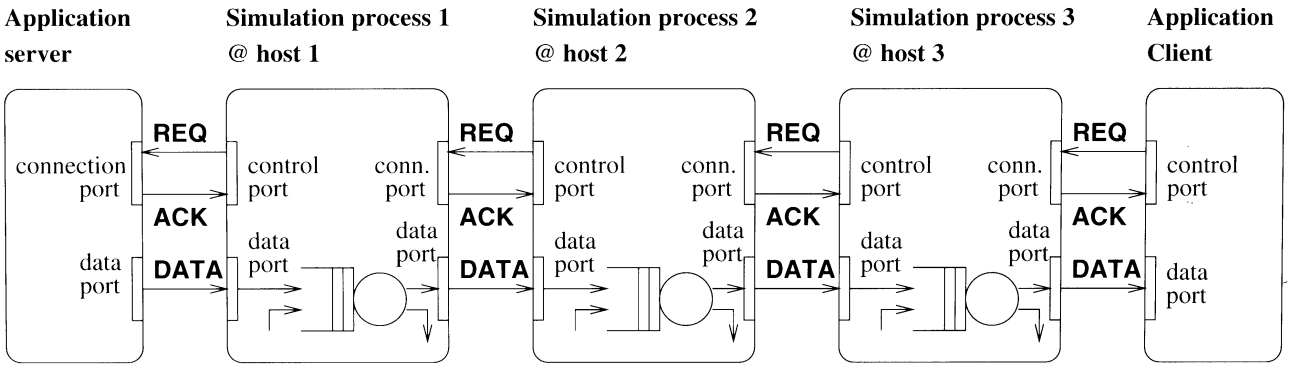

Fig.5– An example of running several simulation process to model the end-to-end path of a networked multimedia application.

the machine name where the upstream process is run and the connection port for the process. These information items are entered by a user when he or she initiates *P*. In Fig. 3, the Connection HOST and Connection PORT entries in the top left block are where the user specifies the information. An implication of this scheme is that the order in which the processes are invoked is from the most upstream one (the application server) to the most downstream one (the application client). When a process starts, it announces its host name and connection port so that the user can specify the information when invoking the immediate downstream process.

Note that the three simulation processes shown in the example in Fig. 5 are run at three different hosts but this is not necessary. Our connection mechanism allows multiple simulation processes running at the same host because UNIX Socket is an inter-process communication mechanism where the communicating processes can be at the same host or at distinct hosts.

We proceed to describe how each simulation process handles a received message. Recall that a message can be either a data or control message. The handling of data has been described in the previous chapter. As for the handling of control messages, we will describe below separately for the request and acknowledgment messages. Note that the messages are encapsulated in datagrams. Hence we will refer to a datagram containing a request (resp., acknowledgment) message a *request* (resp., *acknowledgment*) *packet*.

Request packet.

Upon receiving the packet from its connection port, a simulation process looks up the packet header to find out the IP address and port number of the sender of the packet, which corresponds to the host and control port of the downstream process. The simulation process also reads the first two bytes in the payload portion of the packet and interprets the two bytes as the port number of the data port of the downstream process. The simulation process uses its control port to send the message to the connection port of the upstream process, after altering the first two bytes to its own data port number. In this way, when the upstream process receives the packet, it will know the number of the data port of this process.

Acknowledgment packet.

Upon receiving the packet from its control port, a simulation process uses its connection port to forward the packet to the control port of the downstream process.

Note that as far as the application client is concerned, the procedure of sending out a request packet and eventually receiving an acknowledgment packet is as if the client were directly communicate with the application server.

Finally we remark that we have been looking into generalizing the connection scheme described in this chapter to work for more general

applications. One direction is to design the scheme so that it can be applied to applications based on the Real-time Transport Protocol[8] (RTP). This is an ongoing research issue.

4. Evaluating MPEG Audio Transmission

The project of MPEG audio over the Internet provides us a solid system to test the simulator. We have investigated a few novel techniques to improve the reliability of the transmission. The use of packet-level interleaving was reported in Ref. 9). Our recent work[10] is based on FEC and applies priority encoding transmission[11] to achieve graceful quality degradation if lost data cannot be fully recovered using FEC. Through our experience, we have found the simulator a valuable tool. It is particularly useful in distinguishing the merits of the different transmission techniques.

In conducting our performance evaluation, we first tried to understand how the packet loss ratio, experienced by an audio stream, reacted to changes in simulation parameters. We have conducted numerous experiments. Due to space limitation, here we describe two sets of such experiments. The result of the first set of experiments is depicted in **Fig. 6**. The figure summarizes the experimental results of a set of MPEG audio transmissions where a simulation process was inserted in the data path. The particular MPEG audio stream tested had a data rate of 48 KB/s, fixed packet size of 576 bytes, and duration of 21 seconds. For the simulation process, we fixed the values of buffer capacity and service rate to be 6 KB and 96 KB/s, respectively. We also restricted the process to have only one background stream. Then we selected a set of packet size values (100 bytes, 200 bytes, 400 bytes, 600 bytes, 800 bytes, and 1000 bytes) for the background stream. For each packet size, we varied the arrival rate of the background traffic stream from 44 KB/s to 124 KB/s and recorded how packet loss ratio reacts. Here, for each arrival rate, we conducted 8 simulation experiments and took the average packet loss ratio. Then, using the recorded average loss ratios, we derived a curve which showed the relationship between loss ratio and arrival rate. Note that in Fig. 6, packet size values were used to label their corresponding curves.

A similar set of experiments was conducted and the results were shown in **Fig. 7**. The only difference in this set of experiments was that the background stream had an exponentially distributed packet size.

With the knowledge of how packet loss ratio reacted to simulation parameters, we could control the simulation parameters to achieve a particular packet loss scenario. Note that the packet loss ratio was still a random variable and the (random) distribution of lost packets was still determined by the queueing model. One had to run repetitive experiments to confirm or refute the effectiveness of a proposed transmission scheme. However, the simulator helped very much in focusing the experiments around a particular loss ratio so that we could get insight into how a particular transmission scheme performed under the chosen loss ratio. Note that if instead we had run experiments over the real Internet, this insight would have been extremely difficult to acquire because of the huge dynamic range of packet loss ratios experienced by real Internet transmissions.

We used the simulator to evaluate the effectiveness of using packet-level interleaving and/or FEC to increase the quality of MPEG audio transmission. We consider a scheme *effective* if packet loss results in no distortion in perceptual quality. Our preliminary findings were (*i*) packet-level interleaving alone was effective when packet loss ratio is low (less than 1%), (*ii*) the combination of packet-level interleaving and FEC with 40% redundancy bits was effective when packet loss ratio is mediocre (1-5%), and (*iii*) graceful quality degradation was achievable when packet loss ratio was higher, using a special implementation of FEC that assigned different degrees of redundancy protection to different subband signals.

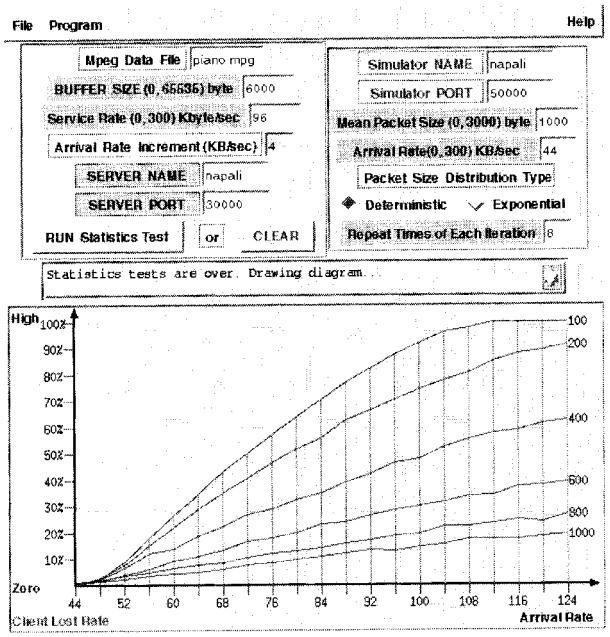

Fig.6– Result of the first set of simulation experiments.

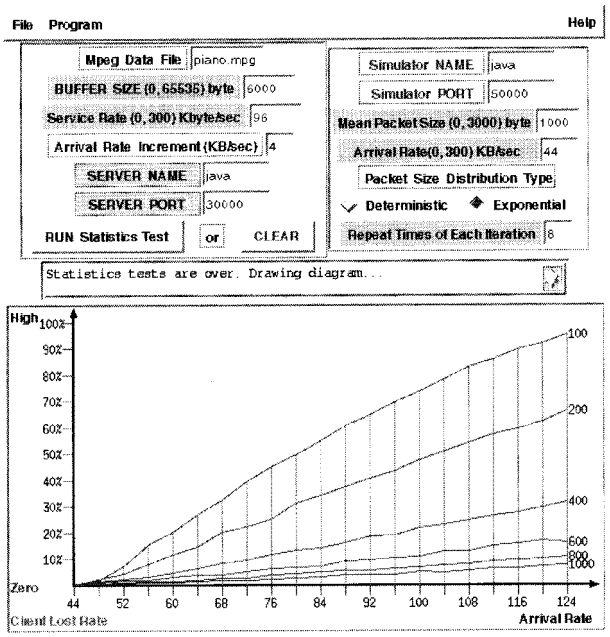

Fig.7– Result of the second set of simulation experiments.

5. Conclusions

In this paper, we described a simulator that uses queueing systems to model the end-to-end paths of networked multimedia. The simulator serves the following purposes.

- It is a performance evaluation tool for the transmission of continuous multimedia such as audio and video.
- It facilitates subjective testing in real time, such as real-time listening of audio.
- It captures essential Internet characteristics including packet loss, packet delay and delay jitter.

The current implementation provides a proof of concept that such a simulator can be built and useful. By no means we declare the work complete. In particular, we have the following directions for improvements.

Aggregation of events related to background streams.

We need to alleviate the impact of timing inaccuracy discussed in the introduction chapter. An approach, which restricts the use of operating system clock timers only to events related to real data packets, was outlined in Chapter 2. We will base on the outline to modify the implementation of the simulator.

Improved event set implementation.

Currently we use a linked list to implement the event set. Insertion of a new event record may become time-consuming when the linked list is long, which happens when there are many packets in the queue or when there are many background streams. To speed up insertion, we can implement the Calendar Queue algorithm described in Ref. 7), as discussed in the end of Chapter 2.

General background traffic types.

Currently we use Poisson processes for traffic modeling. More general traffic models can be added into the simulator. Candidates include multi-state Markovian modeling[12] (a generalization of Poisson modeling), self-similar (long range dependence) processes,[13),14),15)] and batch Bernoulli processes.[16]

Incorporating Internet traffic traces.

Along with the above analytical models, we can generate background traffic using real Internet traffic traces such as those collected Ref. 17). We can also apply the random

distributions derived from real Internet statistics such as the tcplib work.[18]

Accommodating the testing of RTP-based applications.

RTP[8] is becoming a standard for application level framing.[19] Many applications are being developed based on the protocol. It will be desirable to extend the current connection mechanism to accommodate applications based on RTP, as we discussed in Chapter 3.

References

1) Ahn, J.S., Danzig, P.B., Liu, Z. and Yan, L.: Evaluation of TCP Vegas: Emulation and Experiment. Proceedings of ACM SIGCOMM'94, Cambridge, Massachuttes, USA. *Computer Communications Review* **25**, 4, pp. 185-195, (October 1990).
2) Kamada, J., Yuhara, M. and Ono, E.: User-level Realtime Scheduler Exploiting Kernel-level Fixed Priority Scheduler. Multimedia Japan '96, March, 1996.
3) Bertsekas, D. and Gallager, R.: *Data Networks*. 2nd Ed., Prentice Hall, Englewood Cliffs, NJ, 1992.
4) Kleinrock, L.: *Queueing Systems*. **1**, Wiley, New York, 1975.
5) Chen, Y.-M.: Internet Simulator for Testing Networked Multimedia. Technical Memorandum FLA-MTM96-05, Fujitsu Laboratories of America, Santa Clara, CA, November 1996.
6) Jones, D.W.: An Empirical Comparison of Priority-Queue and Event-Set Implementations. *Communications of the ACM*, **29**, 4, pp. 300-311 (April 1986).
7) Brown, R.: Calendar Queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, **31**, 10, pp. 1220-27 (October 1988).
8) Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V.: RTP: a Transport Protocol for Real-Time Applications. Internet RFC 1989, January 1996.
9) Yao, J.-H., Chen, Y.-M. and Verma, T.: MPEG-Based Audio-on-Demand Experiment for the Internet. Interworking '96, Nara, Japan, October 1996. *Global Information Infrastructure (GII) Evolution*, ed. S. Rao, H. Uose, and J. C. Luetchford, IOS Press, Amsterdam, Netherlands, 1996, pp. 503-511.
10) Chen, Y.-M.: Robust MPEG Audio for the Internet. Technical Memorandum FLA-MTM96-10, Fujitsu Laboratories of America, Santa Clara, CA, March 1997.
11) Albanese, A., Bloemer, J., Edmonds, J., Luby, M. and Sudan, M.: Priority Encoding Transmission". Proceedings of 35th Annual Symposium on Foundations of Computer Science, IEEE Computer Science Press, 1994.
12) Li, S.Q. and Huang, C.L.: Queue Response to Input Correlation Function: Continuous Spectral Analysis. *IEEE/ACM Transactions on Networking*, **1**, 3, pp. 678-692 (June 1993).
13) Beran, J., Sherman, R. and Willinger, W.: Long Range Dependence in Variable Bit Rate Video Traffic". *IEEE Transactions on Communications*, **43**, 3, pp. 1566-79 (February 1995).
14) Leland, W.E., Taqqu, M.S., Willinger, W. and Wilson, D.V.: On the Self-Similar Nature of Ethernet Traffic (Extended Version). *IEEE/ACM Transactions on Networking*, **2**, 1, pp. 1-15 (February 1994).
15) Paxson, V. and Floyd, S.: Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, **3**, 3, pp. 226-244 (June 1995).
16) Bolot, J.-C.: End-to-End Packet Delay and Loss Behavior in the Internet. Proceedings of ACM SIGCOMM'93, San Francisco, CA. *Computer Communications Review*, **23**, 4, pp. 289-299 (September 1993).
17) A National Laboratory for Applied Network Research (NLANR): Internet Information Presentation. http://www.nlanr.net/INFO.
18) Danzig, P.B. and Jamin, S.: *tcplib*: A library of TCP/IP traffic characteristics. USC Tech. Report, USC-CS-91-495 (http://netweb.usc.edu/jamin/tcplib/tcplibtr.ps.Z), October 1991.

19) Clark, D.D. and Tennenhouse, D.L.: Architectural Considerations for a New Generation of Protocols. Proceedings of ACM SIGCOMM'90, Philadelphia, Pennsylvania, USA. *Computer Communications Review*, **20**, 4, pp. 200-208 (September, 1990).

Yao-Min Chen is a Member of Research Staff with Fujitsu Laboratories of America, Santa Clara, California, USA. He received the Ph.D and M.S. degrees in Electrical and Computer Engineering from the University of Texas at Austin, in 1994 and 1991 respectively, and the B.S. degree in Electrical Engineering from National Taiwan University in 1987. In 1995, he conducted post-doctoral research in University of Hawaii at Manoa before joining Fujitsu. His research interests are in communication networks, multimedia, and algorithms.