Retransmission-based error control for scalable streaming media systems

Roger Zimmermann
Kun Fu
Frank Liao
University of Southern California
Integrated Media Systems Center
Los Angeles, California, USA 90089-2561
E-mail: kunfu@usc.edu

Abstract. *Large-scale continuous media (CM) system implementations require scalable servers most likely built from clusters of storage nodes. Across such nodes, random data placement is an attractive alternative to the traditional round-robin striping. One benefit of random placement is that additional nodes can be added with low data-redistribution overhead such that the system remains load balanced. One of the challenges in this environment is the implementation of a retransmission-based error control (RBEC) technique. Because data is randomly placed, a client may not know which server node to ask for a lost packet retransmission. We design and implement three RBEC techniques that utilize the benefits of random data placement in a cluster server environment while enabling a client to efficiently identify the correct server node for lost packet requests. We implement and evaluate our techniques with a one-, two-, four-, and eight-way server cluster and across local and wide-area networks. Our results show the feasibility and effectiveness of our approaches in a real-world environment and also identify one solution as generally superior to the other two.* © *2005 SPIE and IS&T.* [DOI: 10.1117/1.1877524]

This paper is a revision of a paper presented at the SPIE conference on Multimedia Computing and Networking 2003, Jan. 2003, Santa Clara, California. The paper presented there appears in SPIE Proceedings Vol. 5019. See Reference [33].

1 Introduction

Continuous media (CM), such as digital video and audio, greatly exceed the resource demands of traditional data types and require massive amounts of space and bandwidth for their storage and transmission.[1] To achieve the high bandwidth and storage required for multiuser CM servers, multinode clusters of commodity personal computers offer an attractive and cost-effective solution to support many simultaneous display requests. One of the characteristics of CM streams is that they require data to be delivered from the server to a client location at a predetermined rate. This rate may vary over time for streams that have been compressed with a variable bit rate (VBR) media encoder. VBR streams enhance the rendering quality, however, they generate bursty traffic on a packet switched network such as the Internet. This, in turn, can easily lead to packet loss due to congestion. Such data loss adversely affects compressed audio and video streams because much of the temporal or spatial redundancy in the data has already been removed by the compression algorithm. Furthermore, important data such as audio/video synchronization information may get lost, which will introduce artifacts in a stream for longer than a single frame. As a result, it is imperative that as little as possible of a stream's data is lost during the transmission between the server and a client.

We were faced with all these constraints when we implemented our CM prototype system called Yima[2] (see Fig. 1). Yima is based on a multinode cluster architecture. Across such nodes, random data placement is an attractive alternative to the traditional round-robin striping. One benefit of random placement is that server nodes can be added or removed with minimum data-redistribution overhead such that the system remains load balanced.[3] However, it is a challenge to implement a retransmission-based error control technique in such a cluster architecture compared to a single-node environment because data is randomly placed and a client may not know which server node to ask for a lost packet retransmission. In this paper, we detail our design and implementation of an efficient packet recovery algorithm that supports multiple server nodes connected to many client stations. Note that our proposed technique could be combined with the existing error control techniques, such as forward error correction (FEC) and error concealment, to support either unicast or multicast applications.

The rest of the paper is organized as follows. Section 2 surveys the related work in this field. Section 3 then details the challenges in a multinode server environment and our approach to the solution. In Sec. 4, we present our extensive experimental results. Finally, Sec. 5 concludes the paper and discusses future research issues.

2 Related Work

There has been considerable work in error recovery techniques that can be applied to real-time streaming applications.[4,5] As shown in Fig. 2, these techniques can be

Paper 03120 received September 10, 2003; revised manuscript received March 11, 2004; accepted for publication May 25, 2004.
1017-9909/2005/$22.00 © 2005 SPIE and IS&T.

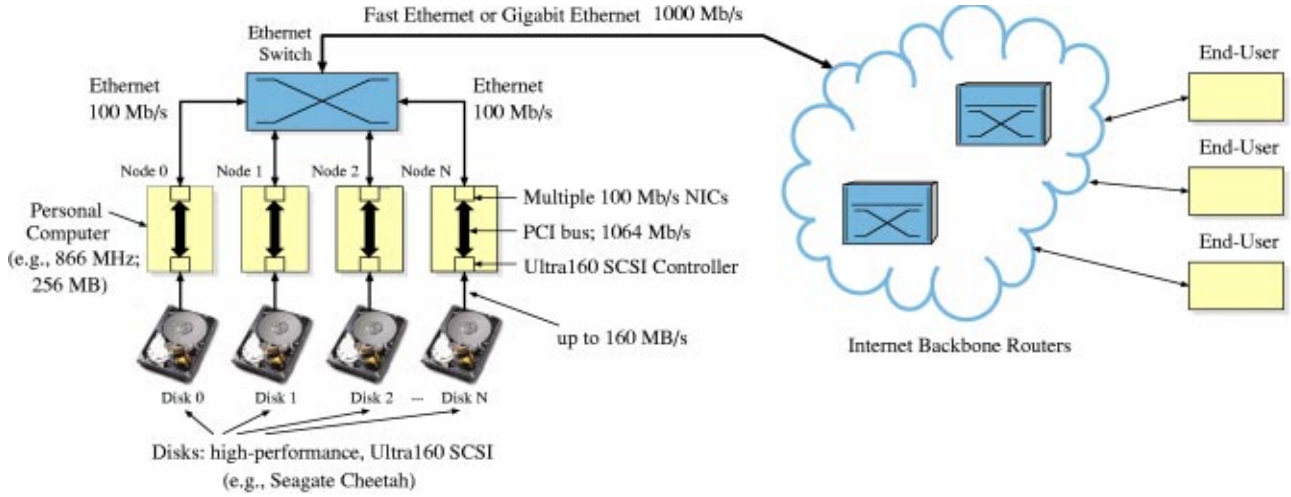


Fig. 1 Multinode Yima continuous media server architecture.

divided into three groups: receiver-based recovery, sender-based recovery, and hybrid error recovery.[6] Receiver-based recovery is also known as error concealment techniques, which rely on the receiver to produce a replacement for the original, lost packet.[7–9] Sender-based recovery requires the participation of the sender and can be further categorized into three groups: retransmission techniques, interleaving techniques, and FEC technique.[10] Here, we review the related work on retransmission techniques.[11–17] A detailed discussion on the other two techniques can be found elsewhere.[4,5]

Retransmission techniques can be distinguished based on their applications: reliable unicast support[12–15] and reliable multicast support.[16,17] For reliable multicast support, Floyd *et al.*[16] proposed a reliable multicast scheme based on retransmission called scalable reliable multicast (SRM). Carle and Biersack[4] provided an overview of existing transport-layer error control techniques and discussed their suitability for use in Internet protocol (IP)-based networks. Perkins *et al.*[5] surveyed packet loss recovery techniques for streaming audio applications operating with IP multicast. Finally, Nonnenmacher *et al.*[17] investigated how FEC can be combined with retransmission to achieve SRM transmissions.

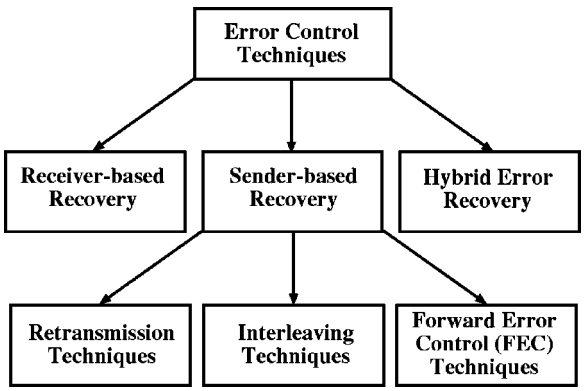


Fig. 2 Classification of error recovery techniques for real-time streaming applications.

For reliable unicast support, previous work has mostly concentrated on analyzing the viability and effectiveness of retransmission-based error control schemes for continuous media applications.[12–15] Marasli *et al.*[12] compared the reliability and delay of sender-based and receiver-based loss detection. Papadopoulos and Parulkar[18] presented a retransmission scheme employing gap-based loss detection. However, their scheme is limited to a single-sender setup as it employs a global sequence number for loss detection. Feamster and Balakrishnan[15] proposed a hybrid packet loss recovery technique that leverages the characteristics of MPEG-4 to selectively retransmit only the most important data if retransmission is possible, otherwise, they rely on error concealment at the receiver.

To our knowledge there has been no proposal so far for retransmission-based error control in an environment where the data is randomly distributed across multiple server nodes. Random data placement enables scale up of the number of nodes in the server cluster with low data-redistribution overhead. However, because of random placement of the data, when a packet is lost, the client cannot determine the correct server node to which it should send a retransmission request (or NACK) only on the basis of the global sequence number as proposed in all the previous work.

3 Approach

For large-scale client-server applications the aggregation of multiple-server machines into a cluster is essential to achieve high performance and scalability. We will first outline our assumed system platform and then describe the challenges and our proposed solution in detail.

3.1 *System Architecture*

Figure 1 shows the overall system architecture of Yima. Our implementation emphasizes the use of low-cost, off-the-shelf, commodity hardware components for the complete end-to-end system. In our prototype implementation, the server consists of a eight-way cluster of rack-mountable Dell PowerEdge 1550 Pentium III 866 MHz PCs with 256 Mbytes of memory running Red Hat Linux 7.0. The media

Table 1 Yima client media type support. All clients are currently implemented on standard Pentium III PC platforms but could also be ported to digital set-top boxes.

Parameters	Client Media Type Support			
Media type	DivX;-) MPEG-4	MPEG-2+DD	MPEG-2 HD	MPEG-1 & 2
Decoder	Software	Creative Dxr2 DVD	Vela Research Cinecast HD	Vela Research Cinecast
Channels	1 video+2 audio	1 video+5.1 audio	1 video+16 audio	4 video+8 audio
Operating system (OS)	Linux (RH 7.*x*)	Linux (RH 7.*x*)	Linux (RH 7.*x*)	Windows NT 4.0
Min. CPU speed	500 MHz	300 MHz	750 MHz	400 MHz
Video resolution	720×480	720×480	1920×1080*i*	4×(720×480)
Audio encoding	MP3	Dolby Digital AC-3	Uncompressed linear PCM[a]	MPEG-1 and 2
Delivery rate	1 Mbits/s	6 to 8 Mbits/s	40+12 Mbits s	4×5 Mbits/s

[a]Our remote media immersion (RMI) system implements a 10.2 channel immersive sound system. See http://imsc.usc.edu/rmi

data is stored on four 18 Gbyte Seagate Cheetah hard disk drives that are connected to the server nodes via Ultral60 small computer standard interface (SCSI) channels.

The nodes in the cluster communicate with each other and send the media data via multiple 100 Mbits/s fast Ethernet connections. Each server is attached to a local Cabletron 6000 switch with a fast Ethernet line. The local switch is connected to both a wide-area network (WAN) backbone (to serve distant clients) and a local-area network (LAN) environment with local clients. Choosing an IP-based network keeps the per-port equipment cost low and is immediately compatible with the public Internet.

The clients are again based on the commodity PC platform. The Yima client software (Yima Presentation Player) runs on either Red Hat Linux or Microsoft Windows. It is structured into several components:

1. The network thread manages both the control and data connections between the servers and the client. The control connection is based on the real-time streaming protocol [RTSP, transmission control protocol (TCP)-based] protocol while the data transmission is carried out via the real-time transport protocol [RTP, used datagram protocol (UDP)-based] protocol.
2. The user interface thread enables user input to be processed such as pause and resume commands.
3. The playback thread retrieves media data that has been stored in the playback circular buffer by the network thread, decodes (i.e., decompresses) it, and renders the resulting data via the appropriate output device (e.g., the sound card for audio or the graphics card for video).

Within this modular architecture we have implemented multiple software and hardware decoders to support various media types. Table 1 lists the different media types that Yima currently recognizes. Our design goal was to not only support the standard MPEG-1, MPEG-2, and MPEG-4 media types at various data rates [i.e., starting from 600 Kb/s for MPEG-4 up to 40 Mb/s for the MPEG-2 high-definition TV (HDTV) format, the standard for which is defined by the Advanced Television Systems Committee, www.atsc.org], but also to enable both constant bit rate (CBR) and VBR transmissions.

3.2 *Server Multinode Design*

An important component of delivering isochronous multimedia over IP networks to end-users and applications is the careful design of a multimedia storage server. The task of such a server is twofold: (1) it must efficiently store the data and (2) it must schedule the retrieval and delivery of the data precisely before it is transmitted over the network. Recall that our server cluster architecture is designed to harness the resources of many nodes and many disk drives per node concurrently. We start by describing the server implementation and then elaborate on the challenges for the media data transmission components.

Because of their high performance and moderate cost, magnetic disk drives have become very popular as storage devices for CM servers. A single high-end disk, such as the Seagate Cheetah X15, can sustain an average transfer rate of more than 30 Mbytes/s (e.g., close to 60 4-Mbits/s streams, under ideal conditions). If—for a large-scale server—a higher bandwidth or more storage space are required than a single disk can deliver, then disk drives are commonly combined into disk arrays.[19] For load-balancing purposes without requiring data replication a multimedia object X is commonly striped into blocks, e.g., $X_0, X_1, \ldots, X_{n-1}$ across an array.[20,21]

Both the display time of a block and its transfer time from the disk are a function of the display requirements of an object and the transfer rate of the disk, respectively. A multimedia object may either require a CBR or a VBR for a smooth display. VBR encoding generally results in a superior visual quality as compared with CBR for the same object size, because bits can be allocated to high-complexity scenes rather than being spread out evenly. However, the bursty nature of VBR media imposes additional challenges for the data scheduling and transmission mechanisms. A CM server should be designed to handle both types of media. Many of today's popular compression algorithms, e.g., MPEG-4, can produce VBR streams.

There are two basic techniques to assign the data blocks to the magnetic disk drives that form the storage system: in a round-robin sequence[22] or in a random manner.[23–25] Traditionally, the round-robin placement utilizes a cycle-based approach to scheduling of resources to guarantee a continuous display, while the random placement utilizes a

deadline-driven approach. In general, the round-robin/cycle-based approach provides high throughput with little wasted bandwidth for video objects that are retrieved sequentially (e.g., a feature length movie). Block retrievals can be scheduled in advance by employing optimized disk scheduling algorithms (such as elevator[26]) during each cycle. Furthermore, the load imposed by a display is distributed evenly across all disks. However, the initial startup latency for an object might be large under heavy load because the disk on which the starting block of the object resides might be busy for several cycles. The random/deadline-driven approach, on the other hand, enables short startup latencies can easily support multimedia applications with nonsequential data access patterns including VBR video or audio, and interactive applications such as 3-D interactive virtual worlds, interactive scientific visualizations, etc. Interestingly, results show that system performance with random data allocation is competitive and sometimes even outperforms traditional data striping techniques for the workloads for which data striping is designed to work best; i.e., streams with sequential access patterns and CBR requirements.[25]

Additionally, a scalable storage architecture should allow for the addition of disks to increase storage capacity and/or bandwidth. By randomly placing data blocks on multiple nodes it is possible to move the minimal number of blocks from an existing storage system to newly added disk drives.[3] For example, increasing a four-disk platform to five disks requires only 20% of all data blocks to be moved, whereas with traditional round-robin striping nearly 100% of all data must be relocated.

Due to its superiority in supporting general workloads, allowing incremental system growth, and providing competitive system performance, we chose random data allocation for our Yima server architecture.

One disadvantage of random data placement is the necessity for a large amount of meta-data: the location of each block X_i must be stored and managed in a centralized repository (e.g., tuples of the form $\langle X_i, \text{disk}_y \rangle$). Yima avoids this overhead by utilizing a pseudo-random block placement. With random number generators, a seed value initiates a sequence of random numbers. Such a sequence is pseudo-random because it can be reproduced if the same seed value is used. By placing blocks in a pseudo-random fashion, the next block in a sequence of blocks can always be found using the pseudo-random number generator and the appropriate seed for that sequence. Hence, Yima needs to store only the seed for each file object instead of locations for every block.

3.3 *Retransmission-Based Error Control*

The Yima cluster architecture takes advantage not only of the distributed storage resources among the multiple nodes, but also of the multiple network connections that link all the nodes together. To avoid traffic bottlenecks, each node transmits the data blocks that it holds directly to the clients via the RTP. Hence, each client will receive RTP data packets from each server node within the cluster. Because RTP packets are connection-less UDP datagrams they might arrive slightly out of order at the client location. Reordering can easily be achieved by using a global sequence number across all packets.

However, an interesting challenge arises when retransmission-based error control is employed. Recall that the current Internet infrastructure provides only best-effort packet delivery and UDP datagrams are not guaranteed to arrive. Therefore, the transmission of CM streams via RTP/UDP requires special provisions if the quality of the rendered streams at the receiving side should be acceptable. One possible solution is the use of FEC. However, FEC always adds a constant percentage of bandwidth overhead irrespective of the network condition. As pointed out by Dempsey *et al.*,[11] if the packet loss rate is very low and timely retransmission can be performed with a high probability of success, a retransmission-based error control (RBEC) approach is an attractive solution since it imposes little overhead on network resources and can be used in conjunction with other error control schemes, such as FEC or error concealment. With Yima we are transmitting some streams that require in excess of 50 Mbits/s bandwidth, for example, for our remote media immersion experiments.[27,28] Our network environment is very stable and usually only a small number of packets are lost during transmission. RBEC has been shown to be an effective solution for CM applications that employ a playout buffer at the client side.[18]

A central question arises when data is randomly stored across multiple server nodes and RBEC is employed. When multiple servers deliver packets that are part of a single stream, and a packet does not arrive, how does the client know which server node attempted to send it?

In other words, it is not obvious where the client should send its request for retransmission of the packet. We have investigated three solutions to this problem. First, the client broadcasts the retransmission request to all server nodes. Second, it uses a heuristic as follows. The server node that successfully transmitted the last packet is a good candidate for a single, targeted retransmission request. Third, additional information is introduced such that the correct server node for the retransmission request can be correctly identified. Note that the idea of adding extra information to a RTP/UDP packet to send the client more detailed information is not new. However, we believe that the fact that it is used to identify a specific sender in a cluster has not been explored in any previous work. One might think of other variations for retransmissions, however, we chose the preceding three schemes because of their simplicity (techniques 1 and 2) or their good performance (technique 3). One important property that we insisted on being preserved across all techniques is that the server should be scalable (i.e., the number of transmission nodes increasable) without requiring any code upgrades at the client sites. Such client-server decoupling is crucial for any real-world, large-scale deployment of video streaming services. It is impractical to require code updates at each of possibly thousands of client sites whenever the server capacity is increased. Hence, we found any technique that required *a priori* knowledge of how data was distributed across the server nodes to be inadequate.

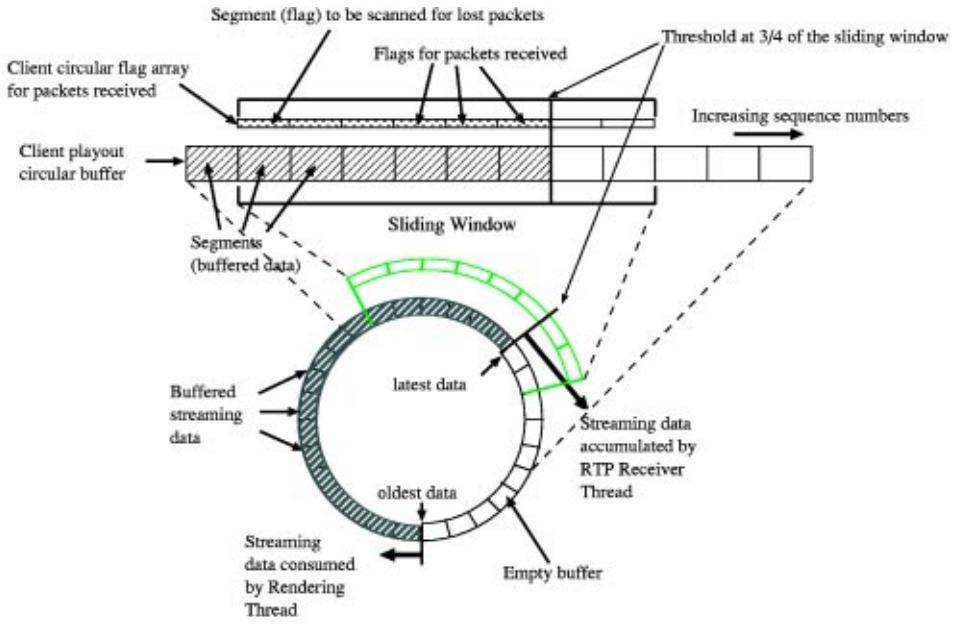


Fig. 3 Design of the client side sliding window mechanism.

3.3.1 *Common implementation features*

We implemented the three chosen techniques within a common framework and we outline the shared components first. Subsequently, we present additional details for each technique.

A client uses a gap-based detection algorithm to initiate retransmission requests. A circular playout buffer is used to accumulate the received RTP packets and a flag array, as shown in Fig. 3, maintains one flag per global packet sequence number (GSN). Each flag is initially set to a ''not received'' status and then updated to ''received'' once the corresponding packet has indeed arrived.

In our experiments, our client circular playout buffer size is 32 Mbytes. Note that the media playout buffer is also used as the retransmission buffer, which is described in details in the next paragraph. However, this buffer size could be smaller and numerous methods have already discussed how to configure the buffer size with RBEC approaches. In these proposed methods, buffer size is estimated based on the network delay measurement,[29] on some stochastic assumptions about the network delay,[11,30,31] or both.[32] (Interested readers could refer to these papers for details.)

Within the circular playout buffer, a sliding window is implemented. The window itself is partitioned into a number of segments of size Q. In our implementation, the sliding window contains 32 segments, i.e., $Q=32$ and each segment contains 32 packets. For every packet, there is a corresponding flag, i.e., a bit in the flag array. Therefore, the flag array contains $Q\times 32=1024$ flags (bits), which map to the 1024 packets in the sliding window. Inside the sliding window, there is a threshold that is set to 3/4 of the window size (i.e., 24 segments). Note that we empirically set the window size and threshold values with the goal of achieving good and stable performance. Finding the optimal values would require the consideration of the round-trip time (RTT), the movie consumption rate, etc., but this has not been the focus of our work. Whenever the client receives a packet with a global sequence number beyond the threshold, the sliding window is advanced forward by one segment. The last segment ''left behind'' is then scanned for gaps in global sequence numbers (i.e., flags that are in the ''not received'' state) that indicate lost packets. For each missing packet a retransmission request is sent to the server(s)—according to one of the three methods—to obtain the missing data.

On receipt of a retransmission request, the server identifies the client via the source IP address of the received request. The server retransmission module maintains a circular buffer per client with the last M previously transmitted packets. Each GSN maps to a particular index in this buffer. The packet corresponding to the sequence number of a particular retransmission request is either still present in the circular buffer, or it has already been replaced by newer packets. If the packet is found, it is sent to the client. Otherwise the request is out of range and no further action is taken.

We implemented this sliding window scheme for several reasons. First, a sequence number gap at the client side usually indicates a packet loss. However, in some cases the packet is just transmitted out of order or delayed due to some temporary condition in the network. For example, Fig. 4 shows the amount of reordering observed during five separate streaming sessions between our server on the East

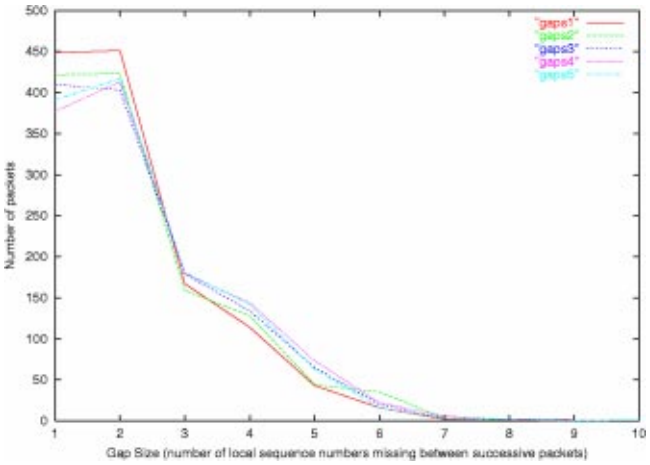


Fig. 4 Frequency distribution of packet sequence number reordering. Gaps of one or two are fairly common, while gaps of more than eight are unusual.

Coast and a client located in our laboratory at the University of Southern California (USC). Reordering gaps of one or two sequence numbers are fairly common, while gaps longer than about eight are very infrequent on this particular path.

Issuing retransmission requests for such reordered packets is obviously unnecessary, and it would waste server resources as well as network bandwidth. Furthermore, the client would receive such a packet twice. This suggests that the client should wait sufficiently long before requesting retransmissions so that it does not make any premature retransmission requests. However, if retransmission requests are delayed too long, the server may no longer hold a copy of the requested GSN in its retransmission buffer. A large number of such dropped requests can potentially make the retransmission protocol ineffective and have a severe effect on the playback quality at the client.

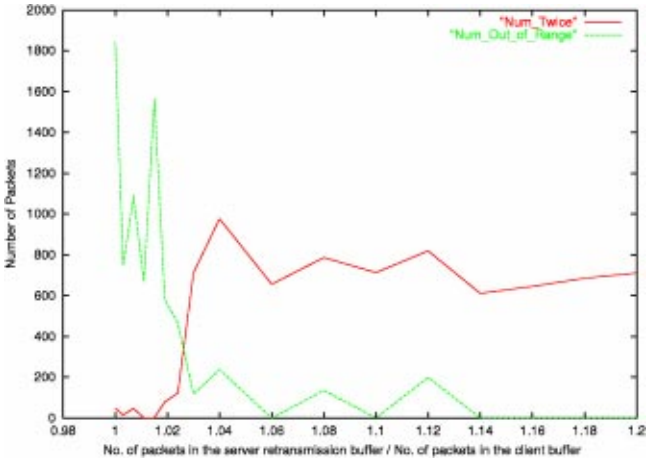


Fig. 5 Number of out-of-range request at the server side and the number of packets received twice by the client as a function of the buffer size ratio on both ends.

Therefore, the correct operation of the described mechanism depends on a useful ratio of $R=M/Q$. Intuitively $M=Q$ should work fine under ideal conditions. Figure 5 shows the total number of out-of-range requests received by the server and the number of packets received twice by the client, as a function of R. Because of the packet round-trip delay, the number of out-of-range packets drops to zero for a ratio slightly larger than 1 ($R \geqslant 1.14$). At the same time, the number of duplicates increases for $R > 1.02$ and remains fairly constant afterward. We conducted all our experiments with this ratio.

The second reason for choosing a sliding window implementation is as follows. Because the window is advanced through incoming packets, the pace of retransmission requests automatically follows the incoming packet stream rate. For a high-bandwidth stream, lost packets are retransmitted more quickly because presumably the data is consumed at a quicker pace. Such an adaptive mechanism is difficult to achieve with timer-based retransmissions. We now describe the differences between the three techniques.

3.3.2 *Broadcast retransmission requests (BCAST technique)*

With the broadcast approach a packet retransmission request is sent to all server nodes. Please note that the request broadcasting in this scenario can be well targeted to include all the server nodes, but no other computers. From observing the RTP/UDP packet header source IP address, the client can easily establish the complete set of server nodes. Once a server receives a request it checks whether it holds the packet, and either ignores the request or performs a retransmission. A disadvantage of this approach is that it wastes network bandwidth and increases server load.

3.3.3 *Unicast retransmission requests (HEUR and GLSN techniques)*

An alternative, more efficient, and scalable method of sending retransmission requests requires that the unique server node that holds the missing packet be identified. This could be accomplished in several ways. For example, the client could reproduce the pseudo-random number sequence that was originally used to place the data across multiple server nodes. This approach has several drawbacks. First, identical algorithms on both the clients and the servers must be used at all times. If the server software is upgraded then all clients must be upgraded immediately too. The logistics of such an undertaking can be daunting if the clients are distributed among thousands of end-users. Second, during scaling operations, the number of server nodes or disk drives changes and hence new parameters must be propagated to the clients immediately. Otherwise, the server nodes will be misidentified. Third, if for any reason the client computation is ahead of or behind the server computation (e.g., the total number of packets received does not match the number of packets sent), then any future computations will be wrong. This could potentially happen if the client has only a limited memory and packets arrive sufficiently out of sequence.

A simple method that decouples the server from the client computation is as follows. As a heuristic, the client

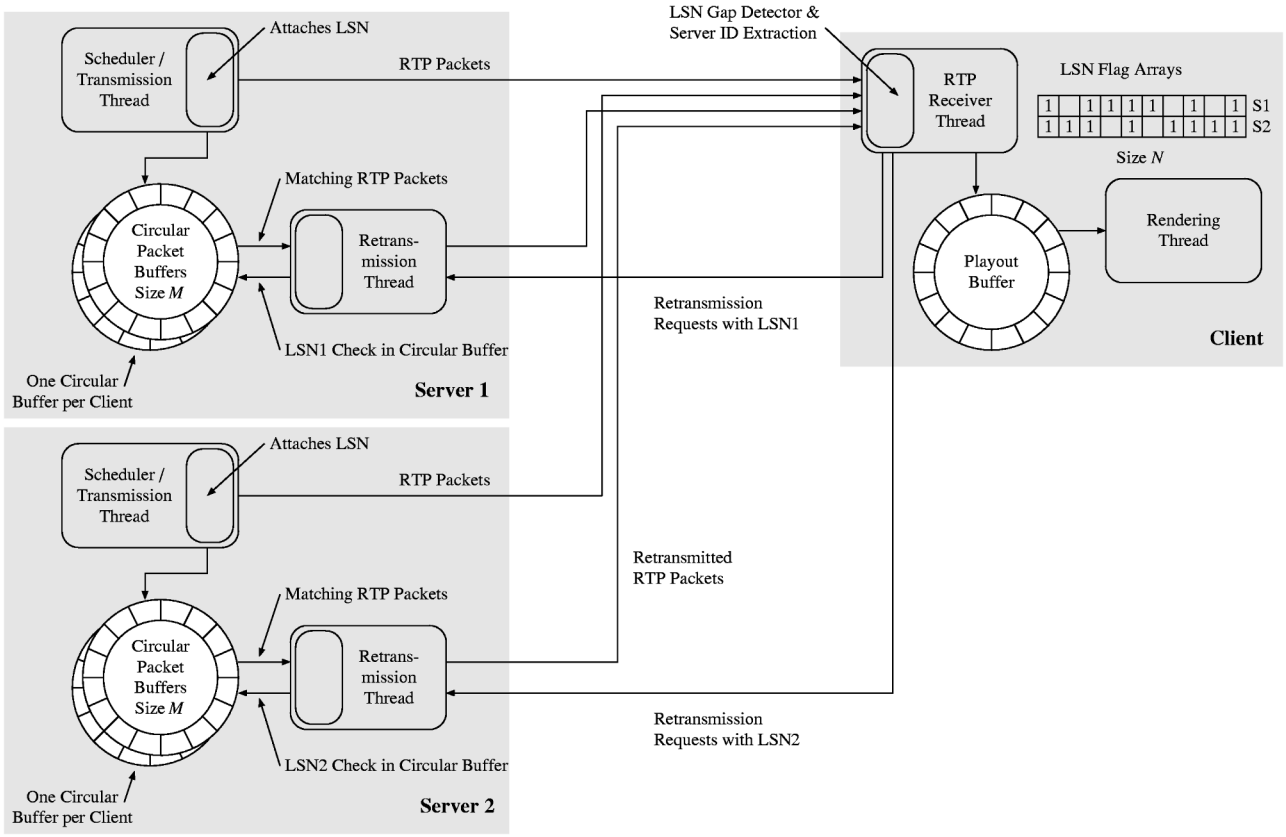


Fig. 6 GLSN implementation: the communication between two server nodes and one client, including the path for retransmission requests.

assumes that each lost packet can be retrieved from the server node that successfully transmitted the last packet prior to the loss. To accomplish this, the client must keep a record of the sender node for each packet received. Then, for each retransmission request, the client retrieves the server IP of the last, prior packet received, and sends the retransmission request only to that node. We refer to this method as HEUR for the rest of this paper. The servers function identically to the nodes in the BCAST scheme.

The misprediction rate of HEUR is affected by the packet loss rate, the number of server nodes, and also by how many or how few packets are sent from each server in sequence. The more packets are sent from each server in sequence, the better the performance of HEUR, because mispredictions occur when packets are lost during the hand-off between two nodes. In our experiments, server storage blocks carry between 500 to 2000 packets each. We investigated the performance of HEUR and document it in a later section.

If the potential for lost packets is not tolerable, then a more robust approach is as follows. The client determines the server node from which a lost RTP packet was intended to be delivered by detecting gaps in node-specific packet sequence numbers. We term these local sequence numbers (LSN) as opposed to the GSN that orders all packets. Although this approach requires packets to contain a node-specific sequence number along with a GSN, the clients require very little computation to identify and locate missing packets. We refer to this addressing approach as global-local sequence numbers (GLSN) for the remainder of this paper. We previously proposed this technique,[33] but presented only limited experimental results at the time.

Next, we describe how the local and global sequence numbers of the GLSN scheme are implemented on both the server and client sides. Subsequently, we introduce an analytical analysis of the misdirected retransmission requests that should be expected with the HEUR technique. Finally, in Sec. 4 we present an elaborate set of implementation test results that show the strengths and weaknesses of each approach.

3.4 *GLSN Implementation*

Figure 6 illustrates the concept of the local sequence numbers with a two-node server. The transmission module at each server node adds a local sequence number to the RTP header of each packet. The LSNs are 32-bit wide, i.e., they wrap around to zero after every set of 2^{32} packets. Furthermore, the LSNs for different client sessions are independent.

As a client starts to receive packets, it acquires the number of server nodes by detecting the number of distinct source IP addresses (it is straightforward to obtain the corresponding IP address of the sender at the receiver by using system calls, such as ''recvfrom()'' on Unix) when receiving the IP packets that contain RTP/UDP payloads. Additionally, the client also maintains an array of bit flags to

keep track of the LSNs received from each node. Hence, on receipt of a new packet, the client first examines the source IP address to identify the server node. Then it sets the corresponding bit for the received LSN in the flag array for that node.

Unlike the BCAST and HEUR techniques, the GLSN client implementation maintains one flag array per server node. Each flag array uses the previously described sliding window mechanism and they operate independently from each other. For our experiments, the total number of flags in all arrays was kept equal for the three techniques to achieve a fair comparison.

3.5 *Analytical Analysis of HEUR Technique*

Any packets lost with either the BCAST or the GLSN technique are due to the loss characteristics of the network. However, with the HEUR approach, additional data may be lost because the client misidentifies the server node being responsible for the lost packets. This phenomenon occurs when the first one or more packets of a movie block are lost. Because of the random assignment of movie blocks to server nodes, the HEUR technique will incorrectly identify the previous server as the sender (because previous packets were sent from there). In this section, we analytically compute the additional number of packets lost that is to be expected with HEUR as compared to either BCAST and GLSN. Table 2 summarizes the analytical terms used in this manuscript.

Let N_R denote the number of retransmission requests during a movie playback, and N_S represents the number of server nodes. We make the following assumptions. Only one retransmission attempt is initiated and every packet including retransmission request, transmitted and retransmitted packet has the same probability of being lost during transmission. (Note that this assumption may not be true in real networks, where the packets are usually lost in bursts. And the bursty nature is often characterized by the Gilbert loss model commonly used in network modeling. More details of the loss modeling are given later in Sec. 4.) The packet loss probability is denoted as p. Then, let L_i denote the probability of losing the ith packet of each movie block and Eq. (1) illustrates how to compute L_i. (We assume that the retransmission requests are always initiated early enough and the client playout buffer size is big enough such that the retransmitted packets can be received at the client in time.)

Table 2 List of terms used repeatedly in this study and their respective definitions.

Term	Definition
N_B	Movie Size in number of blocks
S_B	Block Size in number of RTP packets
N_S	Number of server nodes
p	Probability of losing a packet during transmission
L_i	Probability of losing the i'th RTP packet of each movie block
H_i	Probability of losing the i'th RTP packet of each movie block due to HEUR misidentifying the sender
N_{TL}	Total number of lost packet during a movie playback due to HEUR misidentifying the sender
N_R	Number of retransmission requests generated by client during a movie playback
R_{eff}	Client observed packet loss rate with retransmission
P_{raw}	Client observed packet loss rate without retransmission

$$L_i=\begin{cases}\left(\prod_{j=1}^{i-1} L_j\right)\times p\left[\left(1-\frac{1}{N_S}\right)+\frac{1}{N_S}\times p(2-p)\right]+\left(1-\prod_{j=1}^{i-1} L_j\right)\times p^2(2-p) & \text{if } 1<i\leqslant S_B\\ p\left(1-\frac{1}{N_S}\right)+\frac{1}{N_S}\times p^2(2-p) & \text{if } i=1(\text{first packet}).\end{cases} \tag{1}$$

For example, L_1, the probability of losing the first packet of each movie block, can be computed as the combination of two parts: (1) $p(1-1/N_S)$, the probability of the packet being lost because the client misidentifies a server node, and (2) $1/N_S\times p^2(2-p)$, the probability of the packet being lost either because the retransmission request is lost during retransmission or because the retransmitted packet is lost during retransmission. For the other packets in a movie block, L_i can be obtained similarly.

Now, let H_i represent the probability of losing the i'th RTP packet of each movie block due to the incorrect identification of a server. Following a similar reasoning as for Eq. (1), we derive Eq. (2) to compute H_i.

$$H_i=\begin{cases}\left(\prod_{j=1}^{i-1} L_j\right)\times p\left(1-\frac{1}{N_S}\right) & \text{if } 1<i\leqslant S_B\\ p\left(1-\frac{1}{N_S}\right) & \text{if } i=1\end{cases}. \tag{2}$$

Let N_{TL} denote the total number of lost packets during a movie playback due to HEUR misidentifying the sender, and N_B denotes the movie size in the number of movie blocks. Accordingly, on average, $N_B\times H_1$ denotes the number of first packets in all movie blocks that are lost due to misidentification of the server node. Similarly, $N_B\times H_i$ represents the number of i'th packets in all movie blocks that

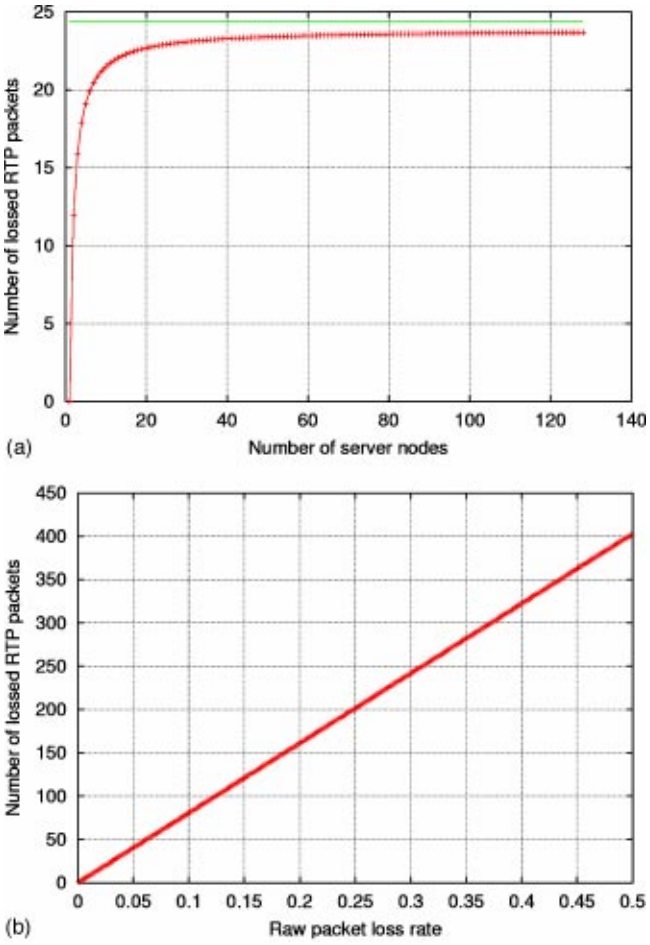


Fig. 7 Additional packet losses caused by misidentifying the server with the HEUR technique (as compared with the GLSN scheme) with $N_B=1074$ and $S_B=2000$.

are lost due to server misidentification. Therefore, we can compute N_{TL} as $N_B\times\Sigma_{i=1}^{S_B}H_i$. Furthermore, based on Eqs. (1) and (2), N_{TL} can be computed as

$$N_{\mathrm{TL}}=N_B\times\sum_{i=1}^{S_B}H_i=N_B\times p\left(1-\frac{1}{N_S}\right)\left(1+\sum_{j=1}^{S_B-1}\prod_{i=1}^{j}L_i\right). \tag{3}$$

Note that N_{TL} quantifies the difference between the HEUR and the GLSN schemes. As an example, consider a system with a loss rate of $p=0.02221$ using $N_S=4$ server nodes. If a movie consists of $N_B=1074$ blocks of $S_B=2000$ packets each, then HEUR will loose an additional $N_{\mathrm{TL}}\approx17.89$ packets during the 25-min playback of this movie.

Figure 7 shows the additional packet losses caused by misidentifying the sender using HEUR compared to GLSN with $N_B=1074$ and $S_B=2000$. Figure 7(a) shows that N_{TL} increases as the number of server nodes N_S increases. This is intuitively understandable because with more server nodes, the HEUR scheme will more likely misidentify the server node to which it will send the retransmission request. Note that the straight line above the curve is the computed theoretical upper bound of N_{TL}. Appendix A provides an

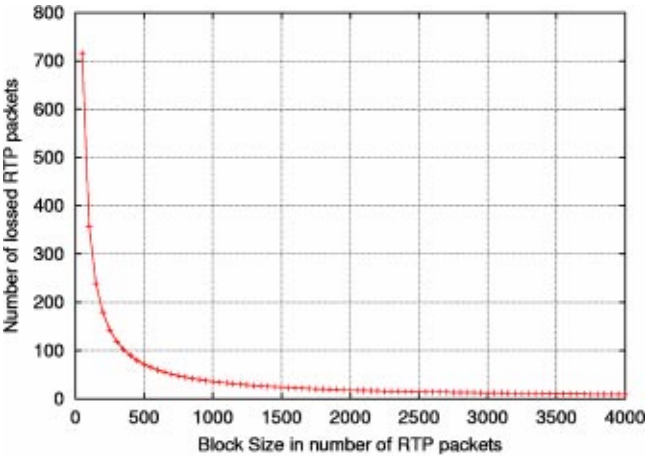


Fig. 8 Additional packet losses due to server misidentification with the HEUR technique (as compared with the GLSN scheme) with different block size S_B and a fixed total movie packet number $S_B\times N_B$ and $N_S=4$.

analysis of this upper bound on N_{TL} when the number of server nodes increases. Figure 7(b) shows the trend of N_{TL} with respect to the increase of the raw packet loss rate p. Note that N_{TL} increases almost linearly as a function of p. Intuitively, this is because as p increases, the number of times that the client needs to decide which server to ask for retransmissions will also increase.

Figure 8 shows the additional packet losses due to server misidentification with the HEUR technique as compared with the GLSN scheme with block size S_B varying from 50 to 4000 and $N_S=4$. Note that the total number of movie packets corresponds to the size of the movie segment *Twister* (see Table 3) we used in our experiments, which is $S_B\times N_B=1074\times2000$. Therefore, by increasing the movie block size S_B, the number of movie blocks N_B decreases. Clearly, the HEUR scheme is affected by the number of consecutive packets from the same server contained in each block of size S_B. With a larger block size S_B, the number of lost packets due to server misidentification is amortized significantly from more than 700 to less than 10.

The results shown in Fig. 8 imply that the HEUR scheme can be adopted when large blocks are used with a streaming server. However, there are several other serious problems with the HEUR scheme. First, the preceding

Table 3 Parameters used in the experiments.

Parameters	Configurations
Test movie *Twister*	MPEG-2 video, AC-3 audio
Average bandwidth	698594 bytes/sec
Length	25 min
Throughput standard deviation	308 283.8
Number of blocks (N_B)	1074
Number of RTP packets in a block (S_B)	2000
Number of server nodes (N_S)	1, 2, 4, 8

Table 4 End-to-end route from one Yima server node (located at USC campus, Los Angeles) to the Yima client (University of Maryland).

Hop Number	Router	RTT (ms)
1	imsc-gw (128.125.163.254)	0.322
2	sal-gw-43 (128.125.3.252)	0.343
3	c2-12008 (128.125.251.65)	0.284
4	ISI-USC.POS.calren2.net (198.32.248.26)	0.750
5	UCLA-ISI.POS.calren2.net (198.32.248.30)	1.348
6	dc-lax-12410a-c2-ucla-pos.cenic.net (137.164.22.56)	1.988
7	hpr-lax-12410-dc.lax-12410a-ge.cenic.net (137.164.22.13)	1.939
8	abilene-LA-hpr-lax-gsr1-10ge.cenic.net (137.164.25.3)	1.897
9	hstnng-losang.abilene.ucaid.edu (198.32.8.22)	33.404
10	atlang-hstnng.abilene.ucaid.edu (198.32.8.34)	46.576
11	atla-atlang.abilene.ucaid.edu (198.32.11.109)	46.624
12	washng-atla.abilene.ucaid.edu (198.32.8.66)	61.843
13	dcne-abilene-oc48.maxgigapop.net (206.196.177.1)	61.811
14	clpk-so3 1-0.maxgigapop.net (206.196.178.46)	62.230
15	206.196.177.126 (206.196.177.126)	62.155
16	Gi3-5.ptx-core-r1.net.umd.edu (129.2.0.233)	62.436
17	Gi5-8.css-core-r1.net.umd.edu (128.8.0.85)	62.369
18	Po1.css-priv-r1.net.umd.edu (128.8.0.14)	62.715
19	128.8.6.139 (128.8.6.139)	62.512
20	imsc.cs.umd.edu (128.8.126.6)	62.869

analysis is based on a simplified assumption that each packet has the same loss probability, which may not hold true in real networks. We show in Sec. 4.2.2 that if the packet losses are very bursty, the HEUR scheme performs much worse than the GLSN scheme. Second, with an increased disk block size, the required buffer size to hold the disk blocks on the server also increases, hence more memory is required. Furthermore, the larger block buffer size also increases the startup latency at the client.[22,25]

4 Performance Evaluation

We implemented and integrated all three of our proposed retransmission-based techniques into our distributed continuous media architecture called Yima, which serves as the platform for assessing the effectiveness of our algorithms. Figure 1 illustrates the overall diagram and the components of our experimental setup. The algorithms are implemented as plug-in modules in both the server and client software. Note that only one retransmission attempt is implemented in the current version of the three techniques, to avoid stalling the real-time media traffic. In all experiments, the servers stream the MPEG-2 movie *Twister* to a client. The list of common experimental parameters and their values is shown in Table 3.

We first evaluate the performance of the GLSN technique in a series of cluster scale-up experiments, and then we compare it with the two other proposed techniques based on the broadcast (BCAST) or heuristic (HEUR) models. In the following sections, we report the detailed results.

4.1 *Server Scale Up Experiments*

We conducted the experiments with two different types of networks: (1) a LAN where the server and client are directly connected through a fast-Ethernet switch and the RTT is usually less than 1 ms, and (2) a cross-continental link via a shared Internet link, where the RTT is around 63 ms. Table 4 shows the data route from one of the servers at USC to our client machine located at the University of Maryland.

4.1.1 *LAN experiments*

In our campus LAN environment, we experience very little packet loss. To evaluate our techniques and to emulate network losses, we implemented a loss module for each server. Whenever a server node must send a packet, the loss module decides whether or not to discard the packet. We used a two-state Markov model, also known as the Gilbert model[34] to emulate the bursty packet loss behavior of a long-haul network. This model is characterized by two conditional probabilities p and q, as shown in Fig. 9. The mean arrival and loss probabilities P_{arrival} and P_{loss} can be computed as

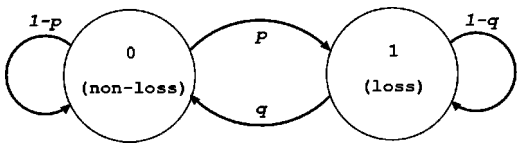


Fig. 9 Gilbert loss model.

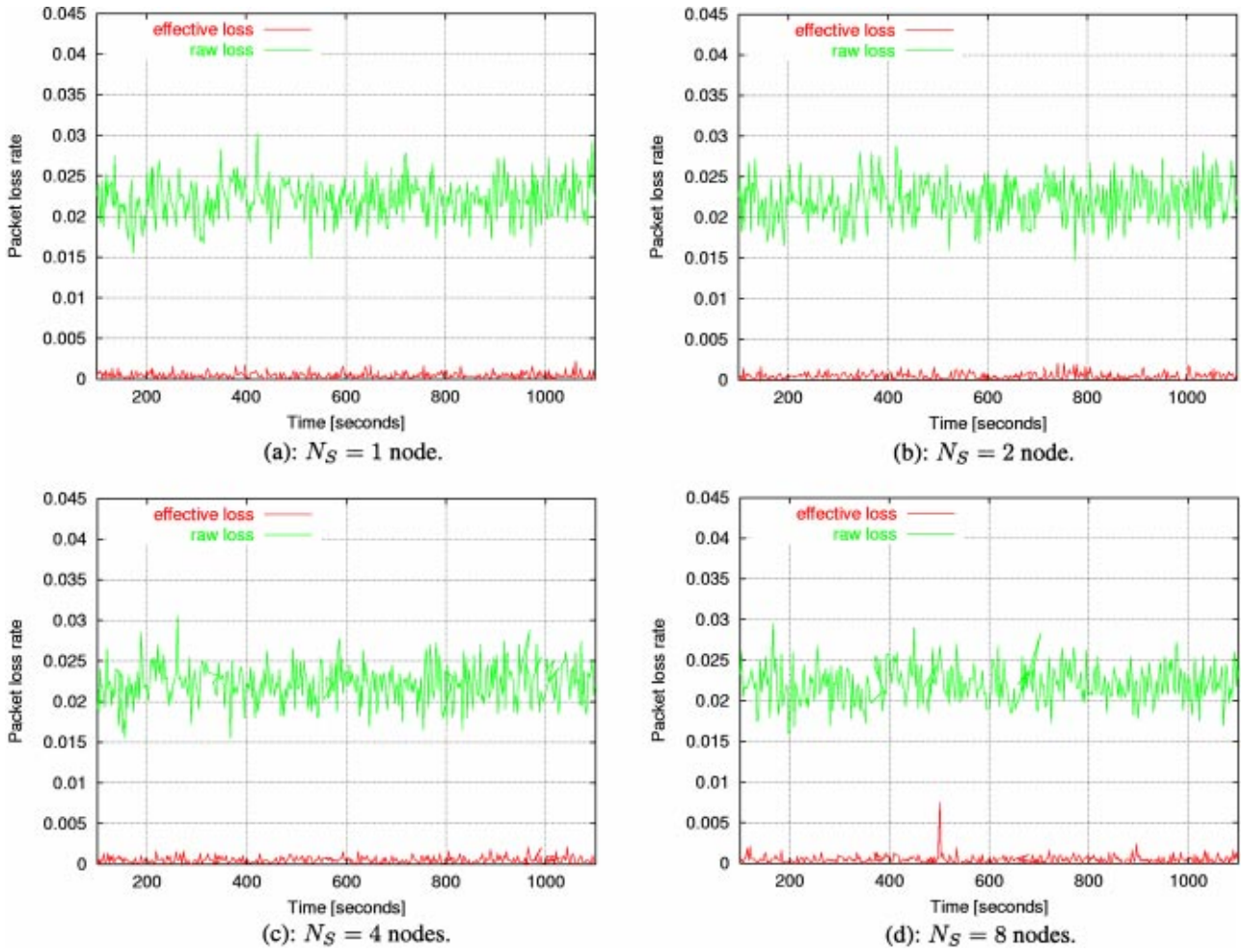


Fig. 10 Effective loss versus raw loss with different numbers of servers in a LAN environment.

$$P_{\text{arrival}} = \frac{q}{p+q}, \quad P_{\text{loss}} = \frac{p}{p+q}. \tag{4}$$

In all our experiments, we set $p = 0.0192$ and $q = 0.8454$ as suggested in Ref. 35. Hence, the resulting mean loss probability P_{loss} is approximately 2.221%.

Figure 10 shows the client observed packet loss rate (termed effective loss) when using the GLSN technique as well as the observed packet loss rate before retransmission recoveries (termed raw loss) with different server configurations ($N_S = 1$, 2, 4, and 8 nodes in a LAN environment). Figure 10(a) shows the raw packet loss rate and effective loss rate measured at the client and generated by the loss model during the streaming of the movie *Twister* in a single-server environment. The packet loss rate decreased from 2.2198 to 0.0461%. Similar results are shown in Figs. 10(b), 10(c), and 10(d), which present the effective loss during 1400 s of the same movie with an increasing number of server nodes. The average packet loss rate declines dramatically from 2.2168 to 0.0481% for $N_S = 2$, from 2.2185 to 0.0512% for $N_S = 4$, and from 2.2187 to 0.049% for $N_S = 8$.

Note that the average raw packet loss rates are always approximately 2.22%, which matches well with the P_{loss} predicted by the Gilbert model. Specifically, with p denoting the raw RTP packet loss rate and q denoting the retransmission request loss rate, while considering only one retransmission attempt, then the effective loss rate P_{eff} can be computed as

$$P_{\text{eff}} = p \times \{q + [(1-p) \times p]\}. \tag{5}$$

If we assume $p = q$, then Eq. (5) can be simplified as

$$P_{\text{eff}} = p^2 \times (2-p). \tag{6}$$

In our LAN experiments, we implemented a loss model for traffic from the server to the client, but not in the other direction, and therefore $p \approx 2.221\%$ and $q = 0$. Based on Eq. (6), P_{eff} is expected to be approximately 0.04933%, which matches well with our LAN experimental results.

Figure 11 shows the RTP packet GSNs at the client side between 100 and 200 s of the movie playback time for one-, two-, four-, and eight-node server configurations. Different colors are used to distinguish the RTP packets sent from different server nodes; for example, there are eight colors in Fig. 11(d). Note that each packet uses the RTP standard 16-bit GSN space, so sequence numbers wrap af-

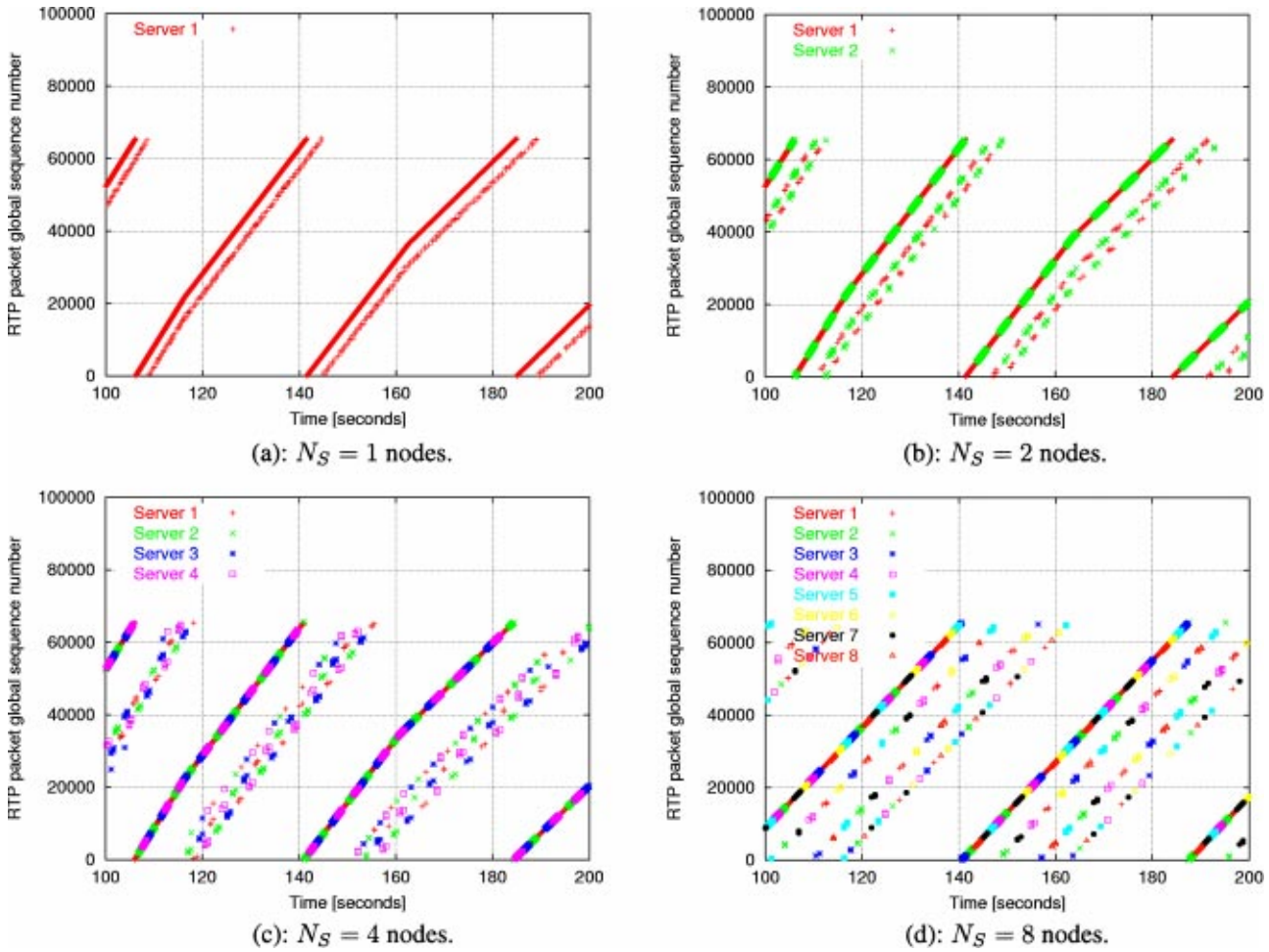


Fig. 11 Client observed RTP packet GSNs for $N_S = 1$, 2, 4, and 8 nodes in a LAN environment.

ter 65,536 packets. In all cases, there are three wraps during the first 100 s. In Fig. 11(a), four pairs of lines are shown. The first line in each pair represents packets that are successfully transmitted initially, while the second line shows the successfully retransmitted packets. Note that in Figs. 11(b), 11(c), and 11(d), there are multiple lines of retransmitted packets. This is because in our GLSN client implementation, the sliding window for each server is operated independently and hence retransmission requests are triggered at different times. Figure 11 shows that the number of retransmitted packets is much less than the volume of initially sent packets. Recall that in our current implementation, GLSN attempts only one retransmission request.

4.1.2 *WAN experiments*

In our WAN experiments, we performed the same set of experiments as reported for the LAN environment. The servers remained unchanged in our USC campus laboratory, while the client was now located across the continental United States in Maryland. The network path of Table 4 details this cross-country Internet link. All data packets traveled through this shared Internet link and therefore some packet loss occurred naturally. The natural loss rate we measured between servers and the client (the corresponding link is shown in Table 4) was quite low at the time of our experiments. Hence, to better evaluate the performance of our technique, we conducted tests with an added loss model, which is the same as we used in Sec. 4.1.1. However, other links may have much higher packet loss rates, as described in Sec. 4.2.

Similar to Fig. 10, Fig. 12 shows the raw loss and the corresponding effective loss using GLSN, with $N_S = 1$, 2, 4, and 8 server nodes. The average raw loss rate decreased from 2.2151 to 0.0692% for $N_S = 1$, from 2.2159 to 0.0662% for $N_S = 2$, from 2.2221 to 0.0639% for $N_S = 4$, and from 2.2214 to 0.0655% for $N_S = 8$. The total average packet loss rate was still approximately 2.2%, which is surprisingly similar to the P_{loss} generated by the Gilbert model we used in our LAN experiments. However, compared with Fig. 10(a), the WAN results show a bit more burstyness than those generated by only the loss model. We attribute this to the impact of the nature of packet losses in the real network. The effective loss rate is between 0.06 and 0.07%, which is a slightly higher that of the LAN experiments. This is because in the LAN experiments, the Gilbert loss model is implemented only on the server side, and hence the client retransmission requests are not subject to losses. However, in the real network, natural losses happen in both directions. If we consider the raw packet loss rate p and the retransmission request loss rate q to be the same—that is,

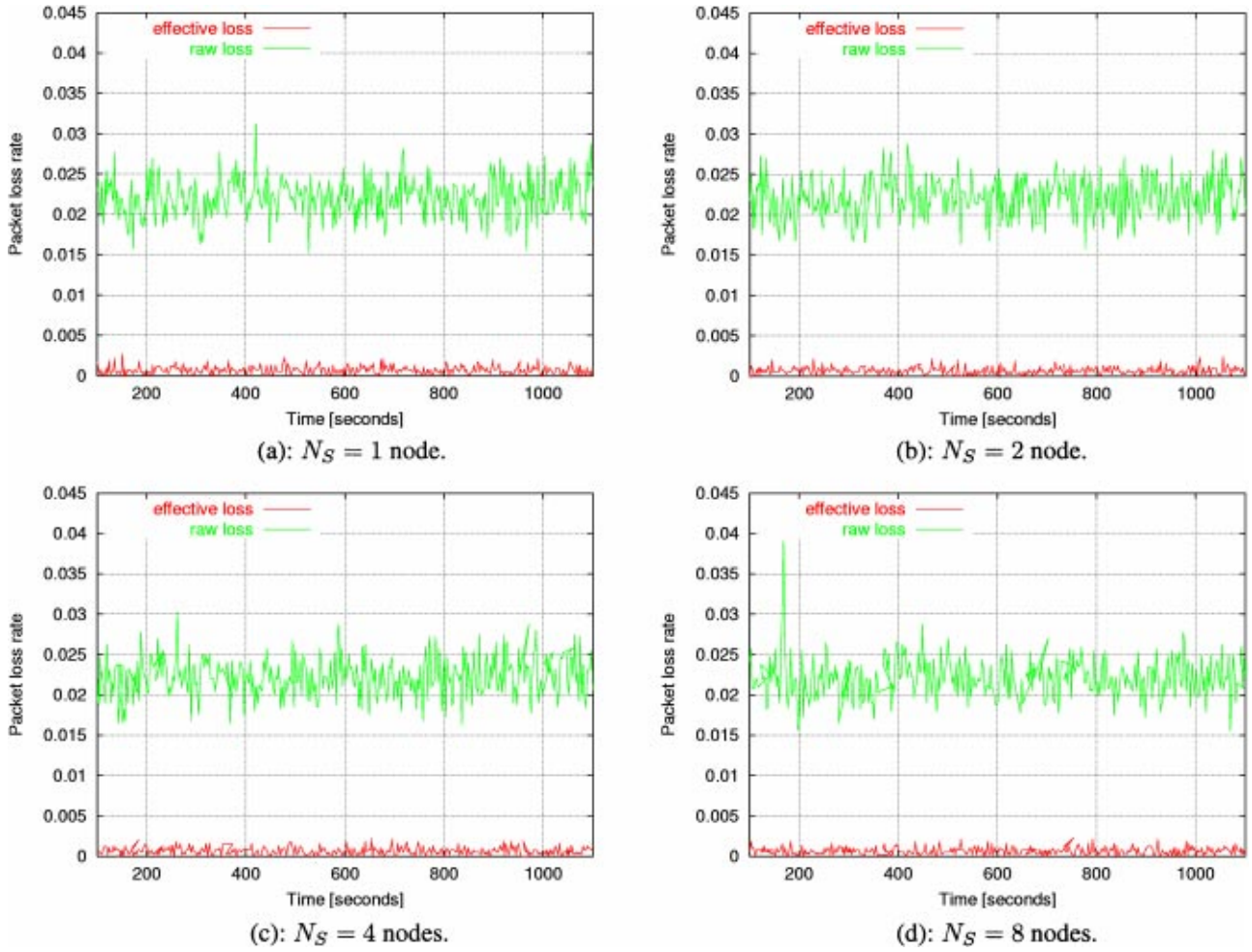


Fig. 12 Effective loss versus raw loss with different number of servers in a WAN environment.

$p = q \approx 2.221\%$ based on Eq. (6)—then we obtain $P_{\text{eff}} \approx 0.0976\%$. Since the WAN experimental results are less than 0.0976%, we infer that the loss rate for retransmission requests is less than 2.221%. Finally, Fig. 13 shows results similar to Fig. 11 for the WAN environment.

4.1.3 *Summary of the scale up experiments*

Table 5 summarizes the results for our node scale up experiments for both LAN and WAN environments. Our experimental results and our detailed discussion in previous sections confirm that the GLSN scheme performs consistently well as the number of server node increases in both LAN and WAN environments.

4.2 *Comparison of Three Techniques*

In this section we compare the performance of the GLSN approach with the BCAST and HEUR techniques that we also implemented in our servers. We conducted experiments with three different networks links: (1) a LAN environment with an RTT delay of less than 1 ms; (2) a cross-continental shared Internet link to the Georgia Institute of Technology in Atlanta, where the RTT is around 54 ms; and (3) another cross-continental link to the New World Symphony in Miami Beach (with an RTT of approximately 73 ms). Table 6 shows the data route from one of the servers at USC to our client machine at GeorgiaTech, and Table 7 shows the data route to our client machine at the New World Symphony.

Note that in all the experiments we use a four-node server configuration, that is, $N_S = 4$. Our comparisons are based on the following two metrics: (1) the effectiveness in improving the raw packet loss rate P_{raw} to the a better effective loss rate P_{eff}, and (2) the overhead as produced by the number of retransmission requests N_R.

4.2.1 *LAN experiments*

Similar to the node scale up LAN experiments, we used the same loss model when we compared the three techniques in the LAN environment. Therefore, the average raw loss rate P_{raw} is expected to be 2.221%. Figure 14 shows the client observed effective packet loss rate P_{eff} and raw packet loss rate P_{raw} with the three techniques, BCAST, GLSN, and HEUR. The results are quite close and on average the packet loss rate drops from $P_{\text{raw}} = 2.2188\%$ to $P_{\text{eff}} = 0.0518\%$ for the BCAST scheme, from $P_{\text{raw}} = 2.2207\%$ to $P_{\text{eff}} = 0.0505\%$ for the GLSN scheme, and from $P_{\text{raw}} = 2.2184\%$ to $P_{\text{eff}} = 0.0536\%$ for the HEUR scheme.

Figure 15 shows the RTP packet GSNs observed at the client between 100 and 200 s of the movie. Figures 15(a)

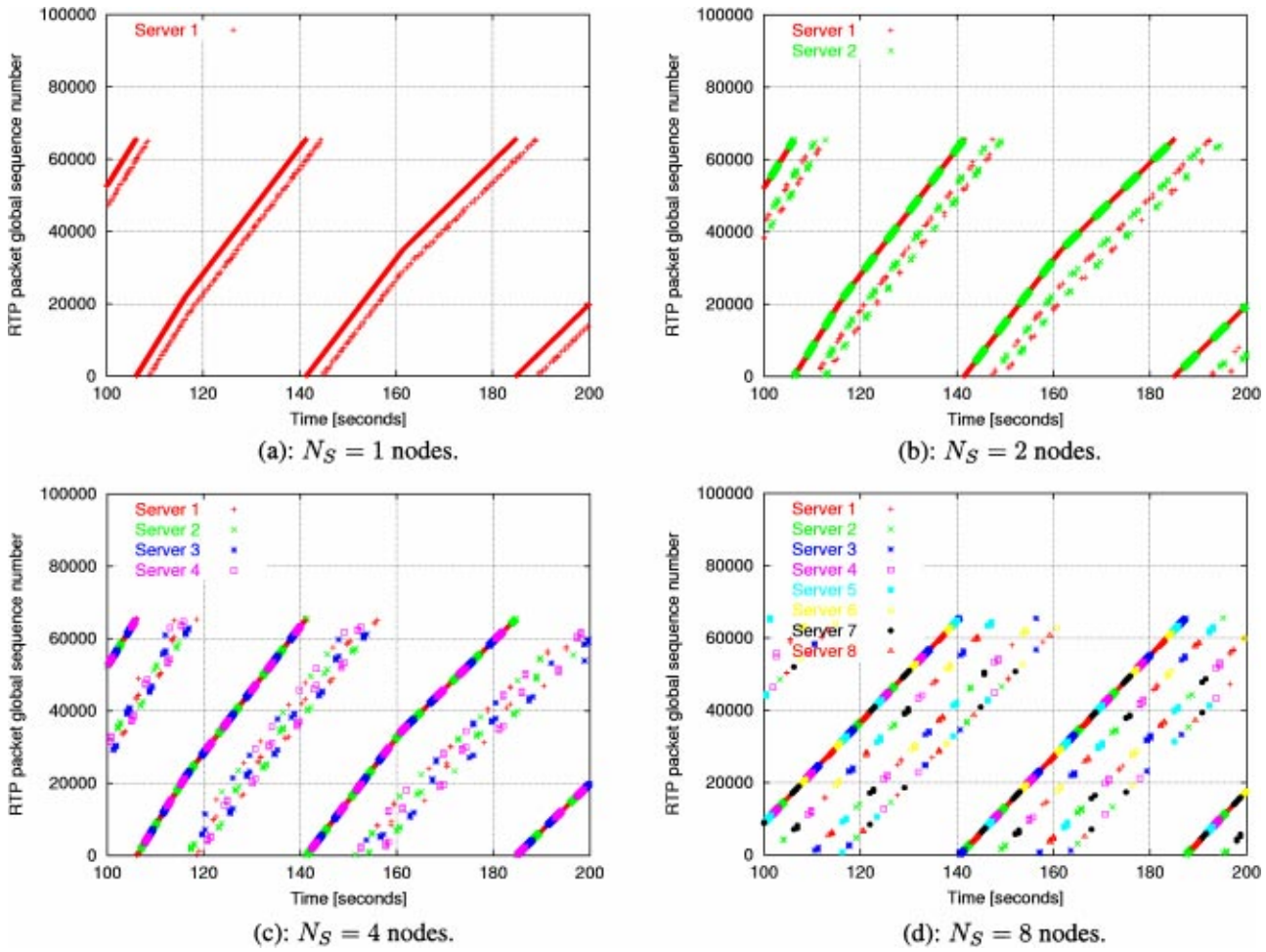


Fig. 13 Client-observed RTP packet GSNs for N_S=1, 2, 4, and 8 nodes in a WAN environment.

and 15(c), which illustrate the BCAST and HEUR techniques, look quite similar. Both of them show three pairs of lines, one for the first-time successfully transmitted RTP packets, and the other for the retransmitted RTP packets. However, the GLSN approach in Fig. 15(b) shows multiple doted lines for the retransmitted packets. This is because in our implementation, there is only one sliding window for the BCAST and the HEUR technique, while there are four independent sliding windows for GLSN with N_S=4 server nodes.

Figure 16 shows the number of retransmission requests that are sent out every second at the client between 100 and 1100 s of the movie. As expected, the GLSN and HEUR results are quite close while the number of retransmission requests for BCAST is almost four times higher. This is also be confirmed by N_R, the total number retransmission requests during the 25 min movie playback. For GLSN, N_R=44,213, for HEUR, N_R=44,152, and for BCAST, N_R =176,712.

Table 5 Summary of measurements results for node scale up experiments for different network environments.

		Number of Server Nodes			
Network Type		N_S=1	N_S=2	N_S=4	N_S=8
LAN	Raw loss[1]	2.2198%	2.2168%	2.2185%	2.2187%
	Eff. loss[2]	0.0461%	0.0481%	0.0512%	0.049%
WAN (Maryland)	Raw loss[1]	2.2151%	2.2159%	2.2221%	2.2214%
	Eff. loss[2]	0.0692%	0.0662%	0.0639%	0.0655%

[1]The average packet loss rate observed at the client without retransmission.

[2]The average packet loss rate observed at the client with retransmission.

Table 6 End-to-end route from one Yima server node (located at USC campus, Los Angeles) to the Yima client (Georgia Institute Technology, Atlanta, Georgia).

Hop Number	Router	RTT (ms)
1	imsc-gw (128.125.163.254)	0.248
2	sal-gw-43 (128.125.3.252)	0.264
3	c2-12008 (128.125.251.65)	0.220
4	ISI-USC.POS.calren2.net (198.32.248.26)	0.843
5	UCLA-ISI.POS.calren2.net (198.32.248.30)	1.302
6	c2-ucla-gsr-dc-lax-12008-atm.cenic.net (137.164.22.5)	2.161
7	hpr-lax-12410-dc-lax-12410a-ge.cenic.net (137.164.22.13)	2.356
8	abilene-LA-hpr-lax-gsrl-10ge.cenic.net (137.164.25.3)	2.252
9	hstnng-losang.abilene.ucaid.edu (198.32.8.22)	33.796
10	atla-hstnng.abilene.ucaid.edu (198.32.8.34)	52.873
11	sox-rtr.abilene.sox.net (199.77.193.9)	52.938
12	gw2-sox.sox.gatech.edu (199.77.194.6)	53.606
13	130.207.251.6 (130.207.251.6)	54.614
14	ucs-imse.cc.gt.atl.ga.us (199.77.128.196)	53.959

Based on the results of these LAN experiments, we make the following observations:

1. Overall, the GLSN technique performs best in terms of both the improvement in the packet loss rate and the minimal additional retransmission overhead.
2. The BCAST technique performs similarly well as GLSN in terms of the improvement in the packet loss rate, however, the retransmission overhead is higher. In our LAN environment there is enough bandwidth and resources available, thus, the additional retransmission request overhead only has little impact on the performance of the system.
3. The HEUR technique also shows fairly good performance in terms of the improvement in the packet loss rate and the retransmission overhead. When considering the packet loss rate, it degrades only approximately 0.0031% compared with GLSN (about 67 RTP packets) during the 25 min movie playback, due to misidentifying packet senders. Based on our analysis in Sec. 3.5, and assuming that each packet has an equal loss probability $p=2.221\%$, we expect $N_{\mathrm{TL}} \approx 17.89$ lost packets. Our measurement result is a somewhat higher than the computed result because the loss model that we employed emulates the busty

Table 7 End-to-end route from one Yima server node (located at USC campus, Los Angeles) to the Yima client (New World Symphony, Miami, Florida).

Hop Number	Router	RTT (ms)
1	imsc-gw (128.125.163.254)	0.264
2	sal-gw-43 (128.125.3.252)	0.306
3	c2-12008 (128.125.251.65)	0.182
4	ISI-USC.POS.calren2.net (198.32.248.26)	0.705
5	UCLA-ISI.POS.calren2.net (198.32.248.30)	1.283
6	c2-ucla-gsr-dc.lax-12008-atm.cenic.net (137.164.22.5)	2.142
7	hpr-lax-12410-dc-lax-12410a-ge.cenic.net (137.164.22.13)	2.476
8	abilene-LA-hpr-lax-gsrl-10ge.cenic.net (137.164.25.3)	2.267
9	hstnng-losang.abilene.ucaid.edu (198.32.8.22)	34.202
10	atla-hstnng.abilene.ucaid.edu (198.32.8.34)	52.874
11	abilene-oc3.ampath.net (198.32.252.253)	71.078
12	juniper-to-gsr.ampath.net (198.32.252.194)	71.508
13	nws.ampath.net (198.32.252.202)	72.320
14	user111.209.42.43.dsli.com (209.42.43.111)	72.360

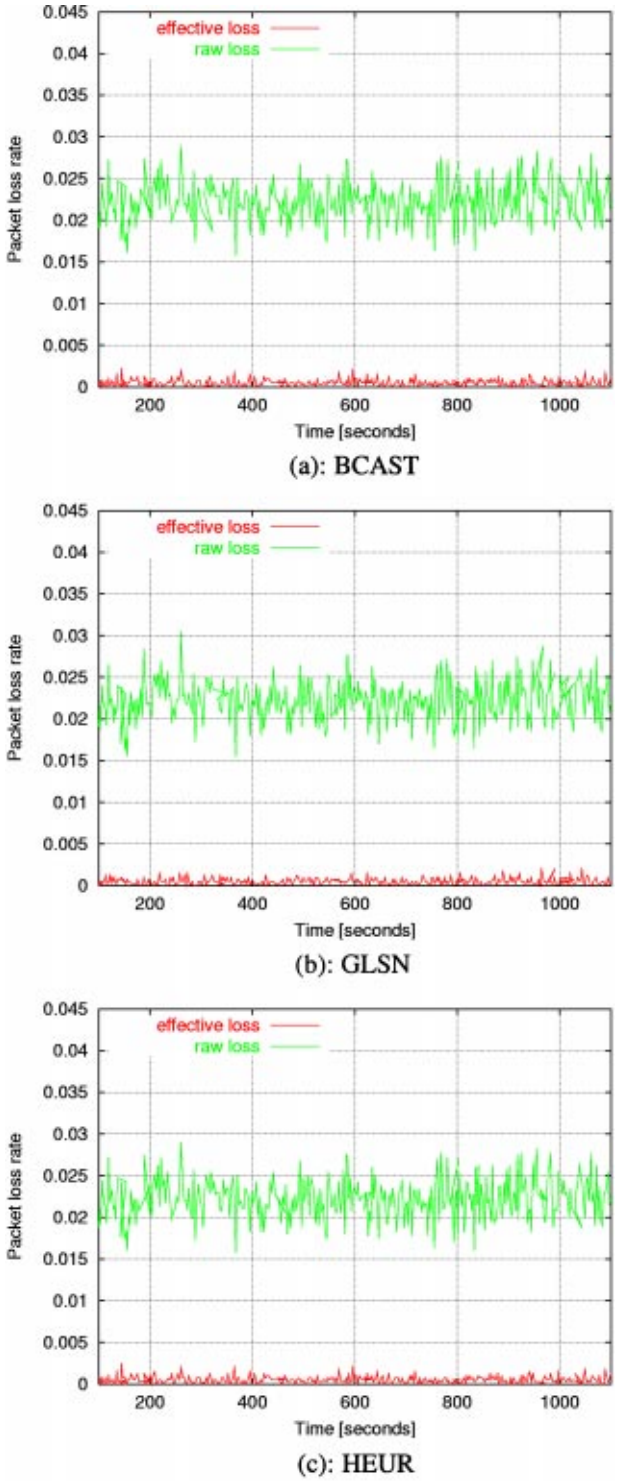


Fig. 14 Effective loss and raw loss for the three techniques with four server nodes in a LAN environment.

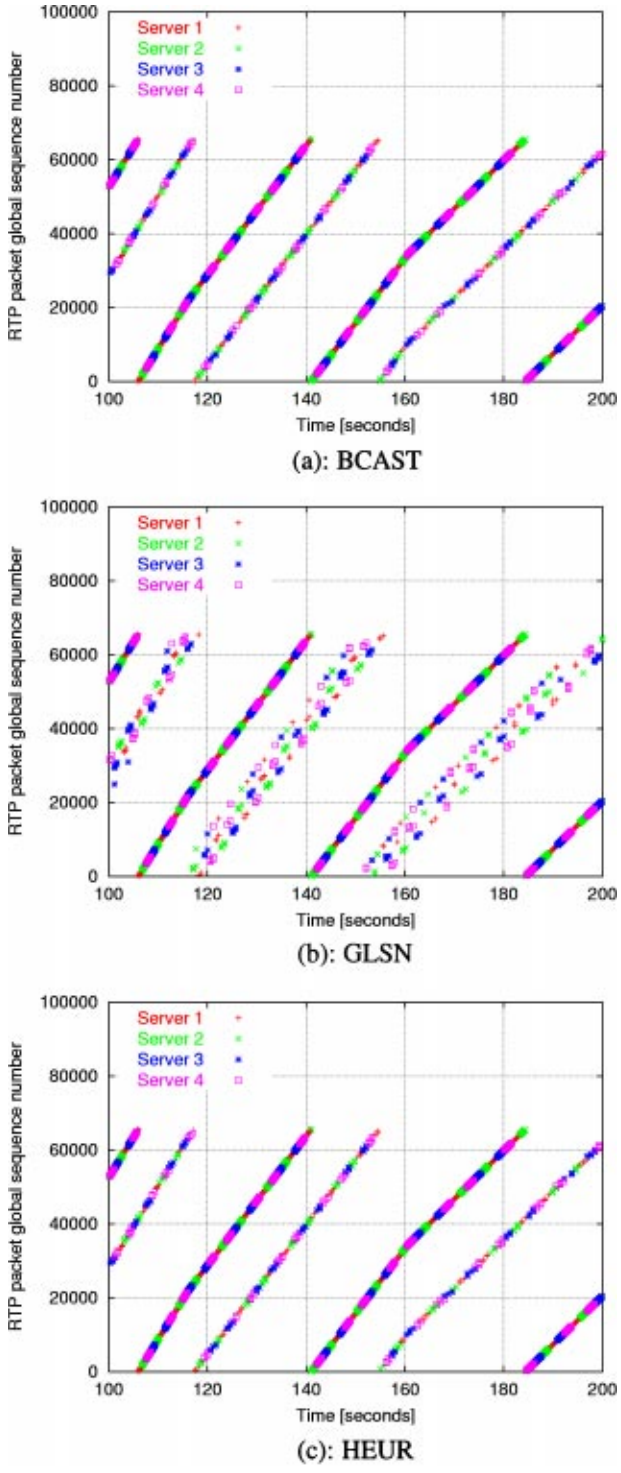


Fig. 15 Client observed RTP packet GSNs for the three techniques with four server nodes in a LAN environment.

nature of real network losses, which means that each packet does not have an equal loss probability. However, since the burstyness is not very high, the HEUR scheme still works reasonably well in this LAN environment.

4.2.2 *WAN experiments*

In our WAN experiments, we performed the same set of experiments as reported for the LAN environment. The servers remained unchanged in our USC campus laboratory, while the client was now located across the continental United States in two different locations, one at the Georgia Institute Technology, Atlanta, and the other at the New World Symphony, Miami Beach, Florida. The network paths are shown in Tables 6 and 7. These are regular, shared Internet links, and therefore some packet losses occurred naturally.

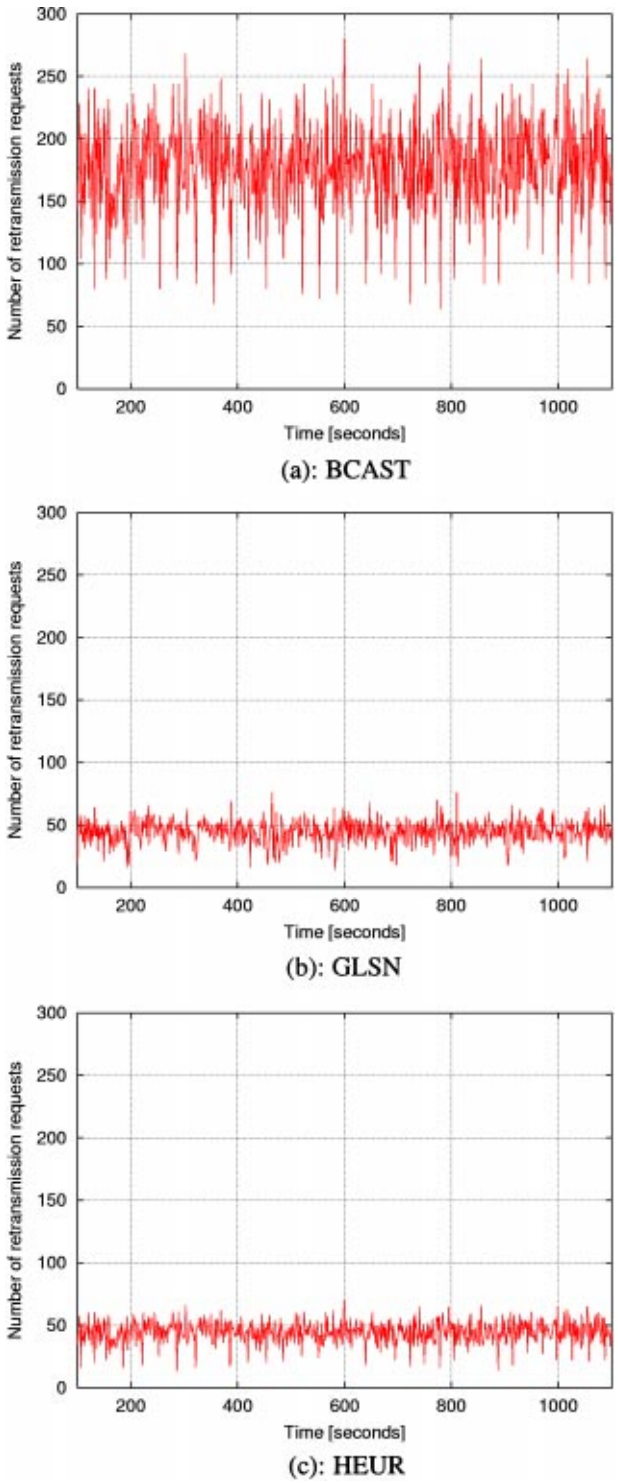


Fig. 16 Number of retransmission requests initiated by a client for three techniques with four server nodes in a LAN environment.

Figure 17 shows the client observed raw packet loss rate P_{raw} and effective packet loss rate P_{eff} between 100 and 1100 s for the BCAST, GLSN, and HEUR techniques under two different WAN environments, i.e., to Atlanta and Miami Beach. For both the Atlanta and Miami Beach links, the average packet loss rate is higher than the average loss rate generated by the Gilbert loss model, which is used in our previous experiments. From Atlanta, the GLSN scheme improves the packet loss rate from 3.7981 to 0.8844%, the BCAST scheme improves the rate from 3.7072 to 2.2158%, and the HEUR scheme improves the rate from 3.5282 to 3.0495%. Among these three techniques, our GLSN approach outperforms the other two. Similar results are obtained from Miami Beach, where GLSN reduced the packet loss rate from 3.6905 to 0.0352%, BCAST reduced the loss rate from 3.7531 to 1.7116%, and HEUR reduced the rate from 4.0185 to 2.8829%. One important observation is that in WAN environments it is very common to see very bursty loss characteristics. Throughout all our tests, the client was occasionally experiencing very long loss bursts. We believe that both the HEUR and BCAST techniques do not work well in such a bursty environment. Recall that we chose a movie block size of $S_B = 2000$ packets on the servers. However, we experienced burst losses much longer than that, sometimes more than 6000 consecutive packets were lost. Hence, the HEUR technique would send retransmission requests to the incorrect server for extended periods of time. For the BCAST scheme, the huge burst losses that happened during a very short interval resulted in a high number of retransmission requests during the same period of time, which would lead to a much higher loss rate for these retransmission requests as compared with the GLSN approach.

Figure 18 illustrates the client observed RTP packet GSNs for all three techniques between 100 and 1100 s of movie playback time. Most of the time there are no packet losses, so there is a single line that wraps periodically. Sometimes, however, there are some long periods of 4000 to 6000 packet losses. With BCAST and GLSN, retransmissions are much more successful than with HEUR.

Figure 19 shows the number of retransmission requests N_R that the client sent to servers between 100 and 1100 s of the movie playback time for BCAST, GLSN, and HEUR, respectively. From Atlanta, the GLSN scheme send out 78,280 retransmission requests, which is similar to the number $N_R = 76{,}456$ generated by HEUR, while for BCAST the number is $N_R = 276{,}324$. For all three schemes, the retransmission requests are sent out in bursts. This is because of the extreme bursty packet loss characteristic of these networks, as we pointed earlier. From both Atlanta and Miami Beach, the retransmission requests are much more bursty for the BCAST approach because the number of requests is multiplied.

Based on the results of our WAN experiments, we make the following observations:

1. It is very common to see extremely long sequences of lost packets. For example, sometimes 5000 or 6000 packets are lost consecutively.
2. In a bursty environment, the GLSN scheme performs much better than either BCAST or HEUR in terms of both the improvement of the packet loss rate and the additional retransmission overhead.
3. The BCAST scheme performs badly because of the high probability of losing retransmission requests due to the much larger quantity of these requests when large number of packets are lost at same time.

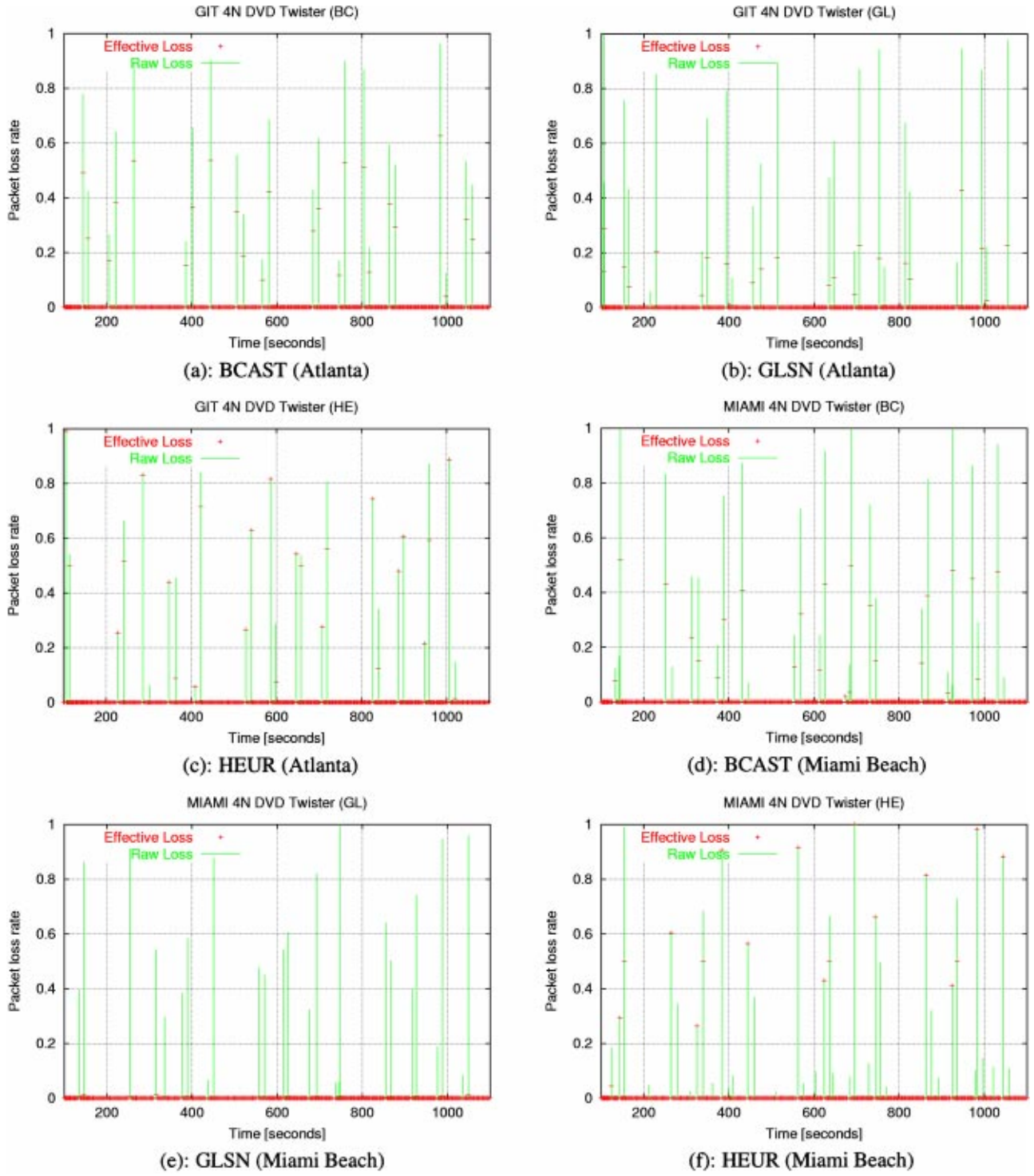


Fig. 17 Effective loss versus raw loss for the three techniques with four server nodes in a WAN environment.

4. The HEUR scheme performs the worst under extremely busty network conditions.

4.2.3 *Overhead measurements*

Table 8 shows the number of sent and received retransmission requests on each server node and client machine in both LAN and WAN environments. We denote the number of correct retransmission requests received by a server during the movie playback time as SR_{corr}. The total number of retransmission requests received by server is denoted as SR_{tot}. We use S_{rcv} to represent the summation of the number of retransmission requests received by serves during the movie playback time. Finally, the percentage of the summation of N_R, the number of retransmission requests generated by the client, is denoted by P_{rcv}.

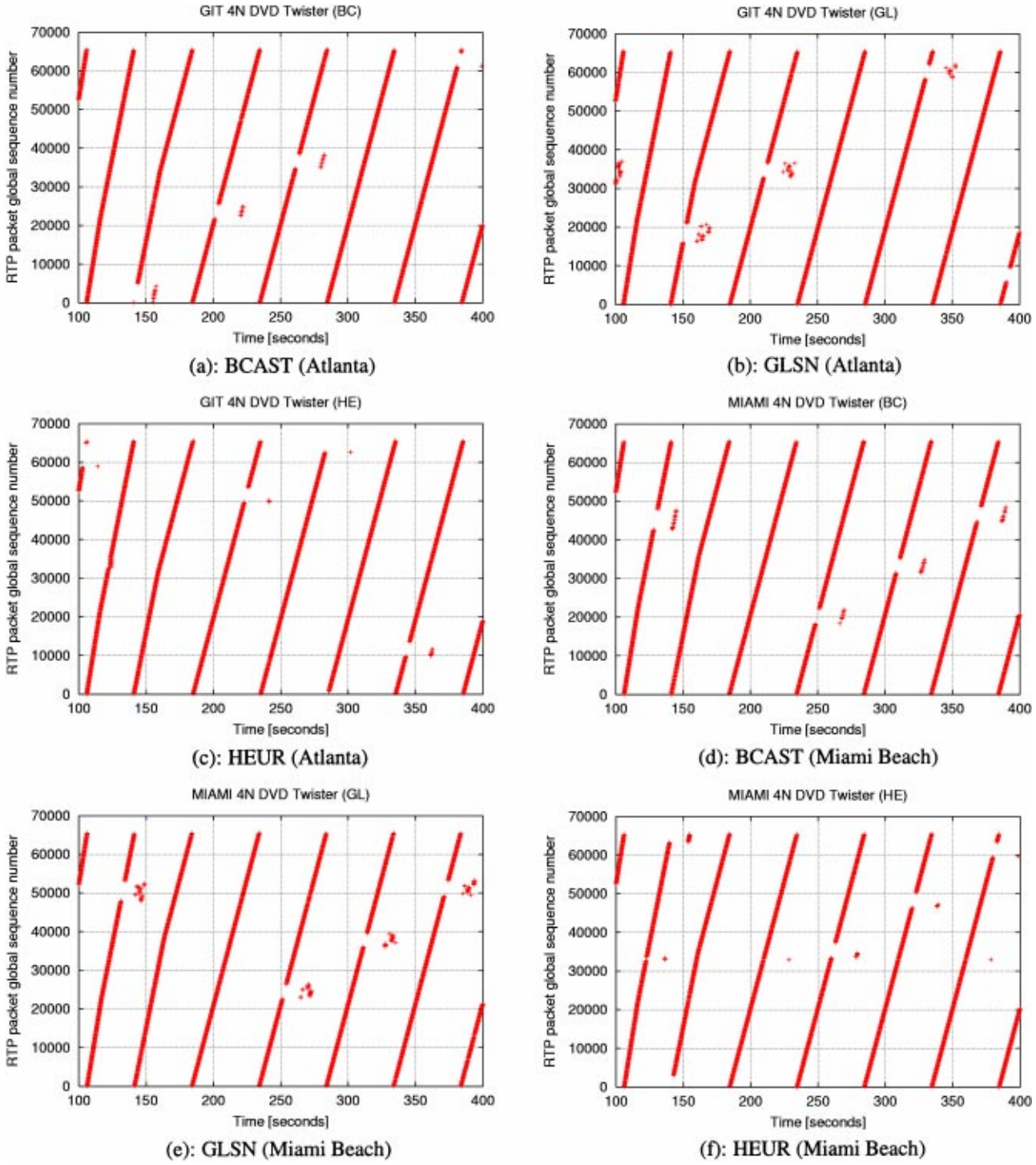


Fig. 18 Client-observed RTP packet GSN for the three techniques with four server nodes in a WAN environment.

For the BCAST scheme in a LAN environment, SR_{corr} and SR_{tot} for all four server nodes are quite close. However, for the Atlanta and Miami Beach links, SR_{tot} for all four server nodes are close, but SR_{corr} are different. When we look at S_{rcv}, the summation of the number retransmission requests received by all servers, SR_{corr} are around 1/4 the SR_{tot} for all environments, LAN, Atlanta, and Miami Beach. For example, for LAN, $SR_{\text{corr}}=44{,}716$ and $SR_{\text{tot}}=178{,}853$; for the Atlanta link, $SR_{\text{corr}}=27{,}737$ and $SR_{\text{tot}}=110{,}801$; and for the Miami Beach link, $SR_{\text{corr}}=41{,}012$ and $SR_{\text{tot}}=163{,}738$. This is confirmed by P_{rcv} as well.

For the HEUR scheme, because the low burstyness in packet loss model as we discussed in previous section, HEUR performs well in LAN experiments, which leads to quite similar SR_{corr} and SR_{tot} values for each server node and across the server nodes. On the contrary, due to the

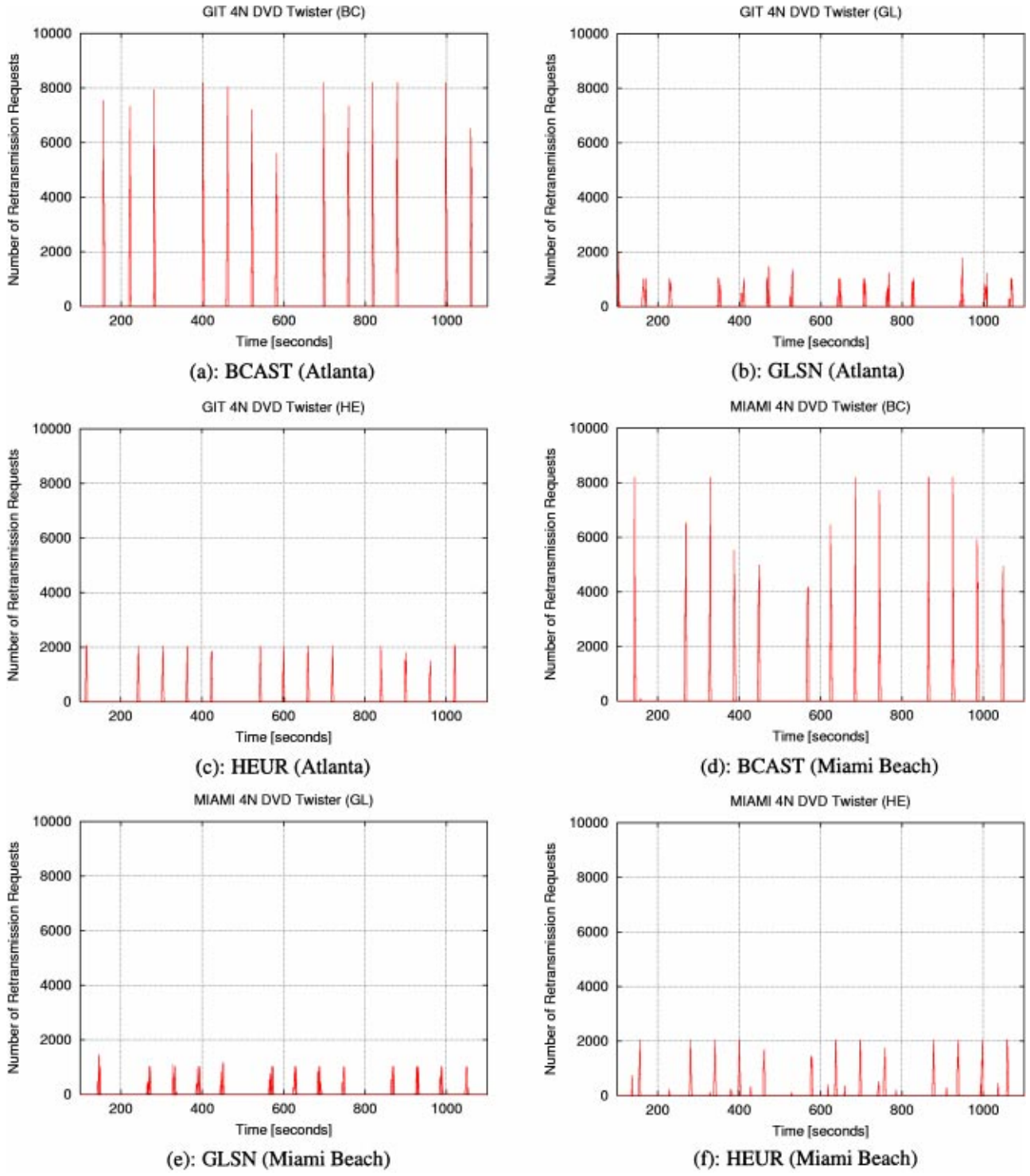


Fig. 19 Number of retransmission requests initiated by a client for the three techniques with four server nodes in a WAN environment.

high bursty loss nature in the Atlanta and Miami Beach links, both SR_{corr} and SR_{tot} are quite different for each server nodes and across server nodes.

We make several interesting observations:

1. In a LAN environment, almost all the retransmission requests are successfully received by the servers, which is illustrated by $P_{rcv}=99.95\%$ for SR_{tot} with BCAST, and $P_{rcv}=99.78\%$ for SR_{tot} with HEUR.
2. For the Miami Beach link and the BCAST scheme, 54.47% retransmission requests are successfully received by the servers, as compared to 99.48% for the HEUR scheme. Because the GLSN scheme will generate retransmission requests at a similar frequency

Table 8 Overhead measurements for different network environments and different retransmission techniques.

Parameters		LAN		Atlanta		Miami Beach	
Scheme	System	SR_{corr}[1]	SR_{tot}[2]	SR_{corr}[1]	SR_{tot}[2]	SR_{corr}[1]	SR_{tot}[2]
BCAST	server 1	11,132	44,686	6,281	27,647	9,863	40,867
	server 2	11,099	44,737	7,877	27,586	12,563	40,948
	server 3	11,238	44,720	6,529	27,768	10,713	40,922
	server 4	11,247	44,710	7,050	27,800	7,873	41,001
	S_{rcv}[3]	44,716	178,853	27,737	110,801	41,012	163,738
	P_{rcv}[4]	24.99%	99.95%	10.04%	40.1%	13.64%	54.47%
	N_R[5]	178,944		276,324		300,588	
HEUR	server 1	11,580	11,587	1,364	4,813	10,393	25,250
	server 2	11,505	11,508	4,803	28,804	4,458	18,797
	server 3	11,734	11,739	1,678	11,377	4,318	17,621
	server 4	11,737	11,743	2,882	8,635	4,614	21,927
	S_{rcv}[3]	46,556	46,577	10,727	53,629	23,783	83,595
	P_{rcv}[4]	99.73%	99.78%	14.03%	70.14%	28.30%	99.48%
	N_R[5]	46,682		76,456		84,033	
GLSN	N_R[5]	44,213		78,280		78,140	

[1]Number of CORRECT retransmission requests received during the movie playback time.

[2]Total number of retransmission requests received during the movie playback time.

[3]Summation of the number of retransmission requests received by servers during the movie playback time.

[4]Percentage of the summation of the number of retransmission requests received by servers.

[5]Number of retransmission requests generated by client during the movie playback time.

under the similar network conditions as the HEUR scheme, it is reasonable to assume that for the GLSN scheme, almost all the retransmission requests are successfully transmitted to the correct server, which is why the average effective packet loss rate for GLSN is extremely low, only 0.0352%.

3. For the Atlanta link, due to the high bursty retransmission requests of BCAST scheme, 40.1% retransmission requests are correctly received by the servers, as compared to 70.14% for the HEUR scheme. Following a similar reasoning as for the Miami Beach case, we can also deduce that for the GLSN scheme approximately 70% of the retransmission requests are successfully transmitted to the servers, which is why the client still observed 0.8844% packet loss rate after retransmissions. Additionally for the BCAST scheme, among those successfully received retransmission requests, only 10.04% are in fact correct requests (correct means that the requests are sent to the right server), as compared to 14.03% for the HEUR scheme. This is why the average effective packet loss rates are still very high, 2.2158% for BCAST and 3.0495% for HEUR.

4.2.4 *Summary of the comparison experiments*

Table 9 summarizes comparison results for the GLSN, BCAST, and HEUR techniques. Based on our experimental results and the detailed discussions in the previous sections, we conclude that our GLSN technique consistently outperforms the other two approaches, BCAST and HEUR, in both LAN and WAN environments.

5 Conclusions and Future Research Directions

We presented the novel challenges that arise when a multinode server cluster that stores data randomly across nodes

Table 9 Experimental results for different network environments and three techniques.

Parameters		Measurements		
Network	Schemes	Raw Loss[1]	Eff. Loss[2]	N_R[3]
LAN	GLSN	2.2207%	0.0505%	44,213
	BCAST	2.2188%	0.0518%	176,712
	HEUR	2.2184%	0.0536%	44,152
GIT	GLSN	3.7981%	0.8844%	78,280
	BCAST	3.7072%	2.2158%	276,324
	HEUR	3.5282%	3.0495%	76,456
MIAMI	GLSN	3.6905%	0.0352%	78,140
	BCAST	3.7531%	1.7116%	300,588
	HEUR	4.0185%	2.8829%	84,033

[1]The average packet loss rate observed at the client without retransmission.

[2]The average packet loss rate observed at the client with retransmission.

[3]The number of retransmission requests during the movie playback time.

is combined with RBEC. We presented an approach based on sequence numbers that are local per node. With this solution retransmission requests can be sent directly to the correct machine. We implemented our technique and evaluated it with an extensive set of experiments across LAN and WAN environments. The results show that the method is feasible and effective. A possible extension of this work will be to enable multiple retransmission requests per packet.

6 Appendix A: Analysis of the Upper Bound of N_{TL}

6.1 *Lemma A.1* $\forall i>0,\ L_i \leqslant p$

Proof. We prove this lemma with the following two cases:

Case 1: When $i=1$.

$$L_1 = p\left(1-\frac{1}{N_S}\right)+\frac{1}{N_S}\times p^2(2-p) \quad [\text{Eq. (1)}]$$
$$= p\left(1-\frac{1}{N_S}\right)+\frac{1}{N_S}\times p \times[1-(1-p)(1-p)] \quad [p(2-p)$$
$$= 1-(1-p)(1-p)] \leqslant p\left(1-\frac{1}{N_S}\right)+\frac{1}{N_S}\times p \quad [1 \geqslant p \leqslant 0 \text{ and } 1-(1-p)(1-p)\leqslant 1] = p.$$

Case 2: When $i\geqslant 2$.

$$L_i = \left(\prod_{j=1}^{i-1} L_j\right)\times p\left[\left(1-\frac{1}{N_S}\right)+\frac{1}{N_S}\times p(2-p)\right] + \left(1-\prod_{j=1}^{i-1} L_j\right)\times p^2(2-p) \quad (1)$$
$$= \left(\prod_{j=1}^{i-1} L_j\right) \times p\left[\left(1-\frac{1}{N_S}\right)+\frac{1}{N_S}[1-(1-p)(1-p)]\right] + \left(1-\prod_{j=1}^{i-1} L_j\right)p[1-(1-p)(1-p)] \quad [p(2-p)$$
$$= 1-(1-p)(1-p)] \leqslant \left(\prod_{j=1}^{i-1} L_j\right)p\left[\left(1-\frac{1}{N_S}\right)+\frac{1}{N_S}\right] + \left(1-\prod_{j=1}^{i-1} L_j\right)p \quad [1 \geqslant p \geqslant 0 \text{ and } 1-(1-p)(1-p)\leqslant 1] = p.$$

6.2 *Lemma A.2*

1. $N_{TL} \leqslant N_B \times p\left(1-\frac{1}{N_S}\right)\times\frac{p^{S_B}-1}{p-1}$,

2. $\lim_{N_S\to\infty}\left(N_B\times p\left(1-\frac{1}{N_S}\right)\times\frac{p^{S_B}-1}{p-1}\right) = N_B\times p\times\frac{p^{S_B}-1}{p-1}$.

Proof.

1. $$N_{TL} = N_B\times\sum_{i=1}^{S_B} H_i = N_B\times p\left(1-\frac{1}{N_S}\right)\left(1+\sum_{i=1}^{S_B}\prod_{i=1}^{j} L_i\right)$$
$$[\text{Eqn. (3)}]$$
$$\leqslant N_B\times p\left(1-\frac{1}{N_S}\right)\times\left(1+\sum_{j=1}^{S_B-1} p^j\right)$$
$$(\text{Lemma A.1})$$
$$= N_B\times p\left(1-\frac{1}{N_S}\right)\times\frac{p^{S_B-1}}{p-1}.$$

2. Based on the result of Lemma A.2(1), it is straightforward to obtain this.

6.3 *Theorem A.1:* $\forall N_S>0,\ N_{TL}\leqslant N_B\times p\times p^{S_B-1}/p-1$

Proof. Using Lemma A.2, it is straightforward to obtain this theorem.

Based on theorem A.1, we can compute that with $p=0.02221$, $N_B=1074$, and $S_B=2000$, $N_{TL}\leqslant 24.3954$. Note that this is number shown as a straight line in Fig. 7(a).

Acknowledgments

We thank the following students for helping with the implementation of certain Yima components: Vasan N. Sundar, Mehrdad Jahangiri, Rishi Sinha, Sahitya Gupta, Farnoush Banaei-Kashani, and Hong Zhu. This research has been funded in part by National Science Foundation (NSF) grants EEC-9529152 (IMSC ERC), IIS-0082826 and CMS-0219463, and unrestricted cash/equipment gifts from Intel, Hewlett-Packard and the Lord Foundation.

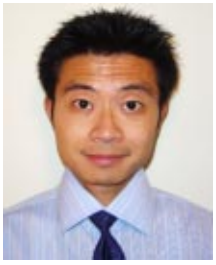
References

1. S. Berson, L. Golubchik, and R. R. Muntz, ''Fault tolerant design of multimedia servers,'' in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 364–375 (1995).
2. C. Shahabi, R. Zimmermann, K. Fu, and S.-Y. D. Yao, ''Yima: a second-generation continuous media server,'' *IEEE Comput.* **35**(6), 56–64 (June 2002).
3. A. Goel, C. Shahabi, S.-Y. D. Yao, and R. Zimmermann, ''SCADDAR: an efficient randomized technique to reorganize continuous media blocks,'' in *Proc. IEEE Computer Society 18th Int. Conf. on Data Engineering (ICDE 2002)*, pp. 1850–1854, San Jose, CA (2002).
4. G. Carle and E. W. Biersack, ''Survey of error recovery techniques for IP-based audio-visual multicast applications,'' *IEEE Net.* **11**(6), 24–36 (1997).
5. C. Perkins, O. Hodson, and V. Hardman, ''A survey of packet loss recovery techniques for streaming audio,'' *IEEE Net.* **12**(5), 40–48 (1998).
6. Y. Wang, A. Ahmaniemi, D. Isherwood, and W. Huang, ''Content-based UEP: a new scheme for packet loss recovery in music streaming,'' in *Proc. 11th ACM Int. Multimedia Conf. (ACM Multimedia 2003)*, pp. 412–421, Berkeley, CA (2003).

7. M. Bystrom, V. Parthasarathy, and J. Modestino, ''Hybrid error concealment schemes for broadcast video transmission over ATM networks,'' *IEEE Trans. Circ. Syst. Video Technol.* **9**(6), 868–881 (1997).
8. B. W. Wah, X. Su, and D. Lin, ''A survey of error-concealment schemes for real-time audio and video transmission over the Internet,'' in *Proc. IEEE Int. Symp. on Multimedia Software Engineering*, pp. 17–24, Taipei, Taiwan (2000).
9. R. Sinha, C. Papadopoulos, and C. Kyriakakis, ''Loss concealment for multi-channel streaming audio,'' in *Proc. 13th ACM Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2003)*, pp. 16–21, Monterey, CA (2003).
10. L. Rizzo, ''Effective erasure codes for reliable computer communication protocols,'' *ACM Comput. Commun. Rev.* **27**, 24–36 (Apr. 1997).
11. B. J. Dempsey, J. Liebeherr, and A. C. Weaver, ''On retransmission-based error control for continuous media traffic in packet-switching networks,'' *Comput. Net. ISDN Syst.* **28**(5), 719–736 (1996).
12. R. Marasli, P. D. Amer, and P. T. Conrad, ''Retransmission-based partially reliable transport service: an analytic model,'' in *Proceeding of IEEE INFOCOMM'96 The Conference on Computer Communications, 15th Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 621–629 (1996).
13. S. Pejhan, M. Schwartz, and D. Anastassiou, ''Error control using retransmission schemes in multicast transport protocols for real-time media,'' *IEEE/ACM Trans. Net.* **4**(3), 413–427 (1996).
14. D. Loguinov and H. Radha, ''On retransmission schemes for real-time streaming in the Internet,'' in *Proceeding of IEEE INFOCOMM'01 The Conference on Computer Communications, 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 1310–1319 (2001).
15. N. Feamster and H. Balakrishnan, ''Packet loss recovery for streaming video,'' in *Proc. 12th Int. Packet Video Workshop*, IEEE Signal Processing Society (Apr. 2002).
16. S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, ''A reliable multicast framework for light-weight sessions and application level framing,'' *IEEE/ACM Trans. Net.* **5**(6), 784–803 (1997).
17. J. Nonnenmacher, E. W. Biersack, and D. Towsley, ''Parity-based loss recovery for reliable multicast transmission,'' *IEEE/ACM Trans. Net.* **6**(4), 349–361 (1998).
18. C. Papadopoulos and G. M. Parulkar, ''Retransmission-based error control for continuous media applications,'' in *Proc. ACM 6th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 1996)*, Zushi, Japan (1996).
19. E. Chang and H. Carcia-Molina, ''Effective memory use in a media server,'' in *Proc. Int. Conf. on Very Large Databases*, pp. 496–505, Athens (1997).
20. V. Polimenis, ''The design of a file system that supports multimedia,'' Technical Report TR-91-020, ICSI (1991).
21. F. Tobagi, J. Pang, R. Baird, and M. Gang, ''Streaming RAID—a disk array management system for video files,'' in *Proc. 1st ACM Conf. on Multimedia*, pp. 393–400, Anaheim, CA, (1993).
22. S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, ''Staggered striping in multimedia information systems,'' in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 79–90, Minneapolis, MN (1994).
23. J. R. Santos and R. R. Muntz, ''Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations,'' in *Proc. ACM Multimedia Conf.*, pp. 303–308, Bristol, UK (1998).
24. R. Muntz, J. Santos, and S. Berson, ''RIO: a real-time multimedia object server,'' *ACM Sigmet. Perform. Eval. Rev.* **25**(2), 29-35 (Sep. 1997).
25. J. R. Santos, R. R. Muntz, and B. Ribeiro-Neto, ''Comparing random data allocation and data striping in multimedia servers,'' in *Proc. ACM SIGMETRICS*, pp. 44-55, Santa Clara, CA (2000).
26. M. Seltzer, P. Chen, and J. Ousterhout, ''Disk scheduling revisited,'' in *Proc. 1990 Winter USENIX Conf.*, pp. 313–324, Usenix Association, Washington, DC (1990).
27. R. Zimmermann, C. Kyriakakis, C. Shahabi, C. Papadopoulos, A. A. Sawchuk, and U. Neumann, ''The remote media immersion system,'' IEEE MultiMedia **11**(2), 48-57 (2004).
28. E. A. Taub, ''On internet of the future, surfers may almost feel the spray,'' *New York Times*, p. C4 of the Circuits section (May 9, 2002); URL:http://dmrl.usc.edu/pubs/NYTimes-RMI.pdf
29. W. Montgomery, ''Techniques for packet voice synchronization,'' *IEEE J. Sel. Areas Commun.* **SAC1**(6), 1022–1028 (1983).
30. G. Barberis and D. Pazzaglia, ''Analysis and design of a packet-voice receiver,'' *IEEE Trans. Commun.* **COMM28**(2), 217–227 (1980).
31. G. Barberis, ''Buffer sizing of a packet-voice receiver,'' *IEEE Trans. Commun.* **COMM29**(2), 152–156 (1981).
32. W. Naylor and L. Kleinrock, ''Stream traffic communication in packet-switched networks: destination buffering considerations,'' *IEEE Trans. Commun.* **COMM30**(12), 2527–2534 (1982).
33. R. Zimmermann, K. Fu, N. Nahata, and C. Shahabi, ''Retransmission-based error control in a many-to-many client-server environment,'' *Proc, SPIE-IS&T Electronic Imaging, Multimedia Computing and Networking 2003*, R. Rajkumai, ed., **5019**, 34–44 (2003).
34. W. Jiang and H. Schulzrinne, ''Modeling of packet loss and delay and their effect on real-time multimedia service quality,'' in *Proc. ACM 10th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2000)*, (2000).
35. M. Yajnik, S. B. Moon, J. F. Kurose, and D. F. Towsley, ''Measurement and modeling of the temporal dependence in packet loss,'' in *Proceeding of IEEE INFOCOMM'99 The Conference on Computer Communications, 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 345–352 (1999).

Roger Zimmermann received his PhD degree from the University of Southern California where he is a research assistant professor of computer science, a research area director with the Integrated Media Systems Center (IMSC), and directs the Data Management Research Laboratory (DMRL). His research interests are streaming media architectures and distributed database integration.

Kun Fu is a doctoral candidate in computer science at the University of Southern California (USC). He did research at the Data Communication Technology Research Institute and National Data Communication Engineering Center in China prior to coming to the United States and is currently a research assistant working on large-scale data stream recording architectures at the Integrated Media System Center (IMSC) and Data Management Research Laboratory (DMRL) in the Computer Science Department at USC. He received his MS degree in engineering science from the University of Toledo. He is a member of the IEEE.



Frank Liao is a multimedia software and content developer, specializing in interactive media, streaming media, and computer graphics. He received his B Arch degree in 1999 from the Southern California Institute of Architecture and his MS degree in 2003 in integrated media systems from the University of Southern California (USC). He has held various positions at Thomas Blurock Architects, USC Institute of Creative Technologies, USC Integrated Media Systems Center, and USC Teaching and Learning Services.