"Scheduled-Multicast" with Application in Multimedia Networks *

Hossam El-Gindy
School of Computer Science and Engineering
University of New South Wales
elgindyh@cse.unsw.edu.au

Chi Nguyen
Department of Computer Science
University of Sydney
chi@cs.usyd.edu.au

Antonios Symvonis
Department of Computer Science
University of Sydney
symvonis@cs.usyd.edu.au

Abstract

We introduce a novel transmission technique, termed "scheduled-multicast". Scheduled-multicast uses existing or additional memory at internal nodes in the transmission paths from the server to the clients to buffer the data stream, and effectively reduce the bandwidth requirements.

We designed a new protocol, Scheduled-Multicast Procotol (SMP), and implemented a prototype system employing SMP to test out the feasibility of scheduled-multicast. The results from our experiments shows that SMP can greatly increase the scalability of the system (i.e., the number of concurrent clients who can be serviced) at a very small cost to the processor utilization. Given the positive results from our prototype, we believe SMP can be used to greatly enhance the scalability of large VoD systems at minimal infrastructure cost.

1 Introduction

Despite recent advances in processor speed, storage capacity and transmission bandwidth, building a cost-effective and scalable continuous multimedia system still remains a challenge. This is best seen in a common application of multimedia systems which is to provide "video-on-demand" (VoD). In such a system, a user can request a particular video clip and have it delivered to the user's multimedia station. The user can perform VCR functions on the clip such as fast-forward, rewind and pause. As a guide, using the common compression standard, MPEG 1, a 90 minute clip requires approximately 1GB of storage and needs to be delivered at a rate of 1.5 Mbit/sec for TV quality playback.

*Work partially supported by Australian Research Council Large Grant A49906214 and an Institutional Grant

To meet the above requirements many technical problems must first be overcame. A common thread running through these problems is the large amount of bandwidth required, both from I/O and the network, and is a major obstacle in building a scalable system.

In this paper we propose a novel technique, *scheduled-mulicast*, to address the network bandwidth problem. The motivation stems from the fact that processing power, disk and memory storage are relatively cheap in comparison with network bandwidth. We believe that by utilizing (and possibly increasing) existing computational resources such as processing power, memory and disk at internal nodes in the path from the server to the users, we can increase the level of concurrency in the system, yet at the same time minimize network traffic. We make an assumption of identical bandwidth requirements for all transmitting streams, but this restriction will be removed in future work.

Our technique is similar to the idea of *multicasting* where rather than sending the same information from the source to each receiver, data packets are copied within the network at fan-out points and sent to multiple receivers. This effectively saves bandwidth in the transmission paths from the source to the fan-out points. Where our technique differs to the traditional multicast is the fact that we introduce a buffer at fan-out points in the network to service requests with different starting times. The scheduling of each transmission is achieved by buffering for the required amount of time such that each client receives its data at its scheduled time (hence the name scheduled-multicast).

The idea behind the scheduled-multicast technique is illustrated in Fig. 1 where four requests for the same movie are made to the video server but with different starting times. In the figure, the arrows represent the network links and the label next to each arrow represents the bandwidth requirement (expressed as the number of transmitting

streams on that particular link). Traditionally, the server will need four transmission streams to service each request as seen in Fig. 1(a). However, if the starting times are known in advance, the bandwidth requirement from the video server to the intermediate router can be reduced to just one stream. This is achieved by buffering data packets from the *leading stream* - the stream with the earliest starting time - at the intermediate router and by forwarding them on at the appropriate time. This is illustrated in Fig. 1(b). The numbers above the slots represent the offsets from the front of the buffer, measured in minutes for simplicity.

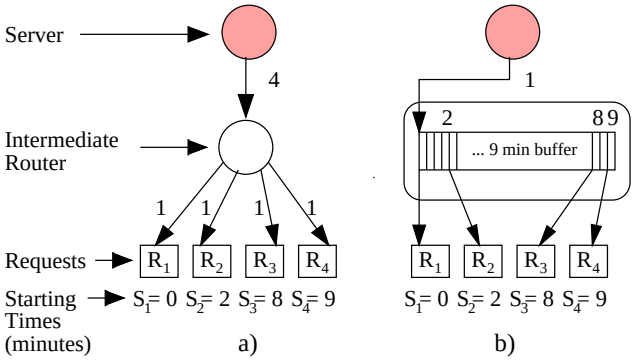

Figure 1. a) Four requests of the same video clip but with different starting times. b) An intermediate router employing a buffer requires only one transmission stream from the server.

The scalability of scheduled-multicast is evident when we extend the simple topology in Fig. 1 to a more realistic topology consisting of many intermediate routers as shown in Fig. 2. In such situations, the video server cannot possibly service all requests individually and late requests must be rejected or delayed until earlier requests have been serviced. However by employing scheduled-multicast, only one stream is needed from the server to the intermediate router in order to service each client.

The idea of scheduled-multicast opens up many interesting research directions. One of these is finding algorithms to determine the best tradeoff for bandwidth and memory utilization. As an example consider Fig. 1. If the bandwidth from the server to the intermediate node is only sufficient for one transmission stream then the only option available is to maintain a buffer of size 9 minutes. However, if the bandwidth available is sufficient for two streams then we have the extra options demonstrated in Fig. 3. By utilizing the extra bandwidth, one can minimize the memory requirement at the router to only 3 minutes (Fig. 3(c). An important question is at what point this tradeoff would make the best use of the system resources.

The other research direction deals with the establishment of a protocol to handle the buffering of streams and their

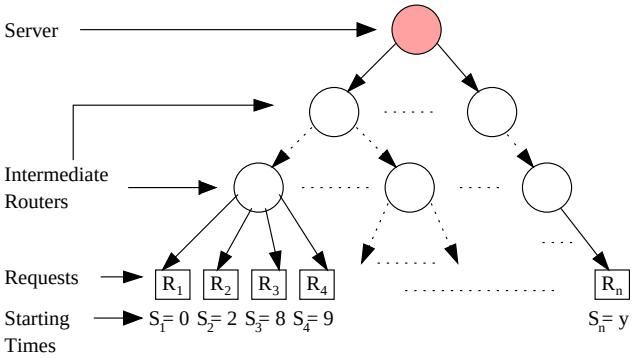

Figure 2. An extended example with a more realistic topology for requests of the same video clip.

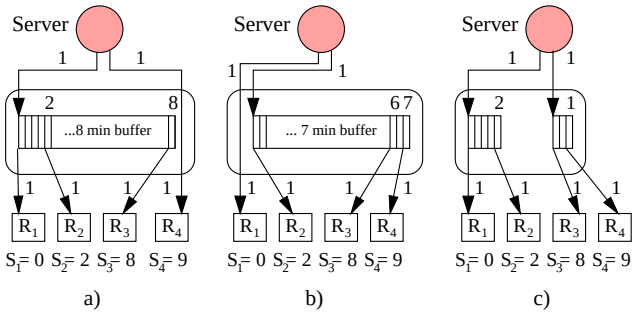

Figure 3. Alternative buffer utilization

transmission at specified times. The protocol must be flexible enough to accommodate for the resource allocation algorithm being employed. For example it must provide a way to reserve a buffer of a specified size. To the best of our knowledge no multicast protocol currently exists which has provisions for the delay of the transmitted streams for the specified amount of time.

In this paper, we focus on the design of the scheduled-multicast protocol. Section 2 examines related work in bandwidth utilization. Section 3 describes our VoD framework and the scheduled-multicast protocol. In Section 4, we describe our implementation of the prototype and provide an analysis of it. Section 5 presents the experimental evaluation of our prototype which indicates that scheduled-multicast greatly increases the number of concurrent users of a VoD system at minimal cost to processor utilization. Finally, Section 6 outlines the future work.

2 Related Work

Previous research into bandwidth management can be separated into two areas: disk I/O bandwidth and network bandwidth. Many of the proposed solutions can be applied to both areas with slight modifications, or in conjunction

with each other.

An important consideration in systems which require a high rate of transfer involving disks is the high latency from disk I/O. By using a *matrix-based allocation scheme*, Özden et al. [13] showed how to avoid the disk latency overhead while minimizing memory buffer space requirements. Their work was extended in [5] by Garofalakis et al. to handle different display rates, retrieval rates and video clip lengths. In addition, they looked at how the matrix-based allocation scheme can be applied to different data layout schemes on disk, namely: *clustering* where entire clip is stored on a single disk, and *striping* where each clip is declustered over all available disks (similar to a RAID file system [14]).

Where multiple requests for the same clip differ by a small time interval, memory can be used to buffer data read from disk, saving subsequent disk accesses. This technique was introduced by Kamath et al. [9] in conjunction with a heuristic for determining when buffer sharing is beneficial. In [15], Shi and Ghandeharizadeh showed that buffer sharing can in fact degrade system performance by exhausting system memory. Instead they propose a more stringent heuristic to calculate the threshold at which sharing is beneficial. Their heuristic takes into consideration the costs of bandwidth and memory, in addition to memory availability, request interarrival times and data rate.

In minimizing network bandwidth consumption, researchers have concentrated on using the multicast technique. The Multicast Backbone network or MBone built on top of the Internet infrastructure is one such example. A survey of IP multicasting and the MBone architecture is presented by McCanne [12]. However, as pointed out by Holbrook and Cheriton [8], current IP multicast has many drawbacks for large scale broadcast of videos in that there is a lack of support for billing, lack of control as to who can send on the multicast channel, and limited IP address space. Instead, they have proposed an extension to the current IP multicast to better cater for multicast channels with multiple subscribers but a single designated source.

The inherent problem of multicast is that all requests must have the same starting time. A simple technique known as *batching* (see [1, 4]) groups requests that have arrived during an interval - called the batching interval - and services them as a whole using a single I/O stream. However, for it to be effective a large batching interval is needed, leading to unsuitable latency time.

A different approach is *stream aggregation* which separates requests into groups by bridging the temporal skew between them. This is achieved through rate adaptation [10, 3] where frames are selectively dropped from the transmitting side and, at the receiver's end, interpolation techniques are used to compensate for the loss of frames. While this technique might be feasible for the image quality of the video stream, the sound degradation is much more noticeable and often beyond the tolerance level. A similar idea to stream aggregation is *adaptive piggybacking* which was proposed by Golubchik, Lui and Muntz [6] and later improved by Aggarwal et al. [2]. With adaptive piggybacking, the bridging of the temporal skew is done by slightly speeding up the display rate of the later stream while at the same time slowing down that for the earlier stream so that they will eventually merge into one stream.

An important point to note is that in most of the work discussed above, the ideas and techniques proposed are orthogonal to our work and as such they can be used in conjunction. For example one can add buffer sharing [9, 15], and rate adaptation [10, 3] to a system employing scheduled-multicast to further enhance system performance.

3 The Scheduled-Multicast Framework

3.1 System Architecture

Our VoD architecture consists of a local server connected to the user Set-Top Box (STB) via a local community network. The local server is connected through the backbone network to remote video servers. The role of the local server is to act as the client's gateway to the remote video servers as well as maintaining bookkeeping information such as costs for each user. It is assumed that the local server has enough resources in terms of bandwidth and processing power to service all the clients connected to it.

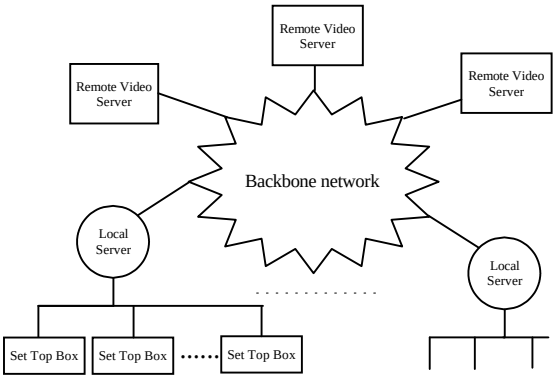

Figure 4. A typical VoD architecture

The backbone network consists of high speed links interconnecting different local servers with the remote video servers. The whole system can be modeled by a connected graph where the nodes represents the intermediate routers and switches of the backbone network, the local servers and the remote servers. The bandwidth on each individual link can be represented by the weight connecting two nodes. For simplicity, bandwidth is measured in units of

number of video streams that can be concurrently transmitted on the particular link. To send the requested clip to the users, a path to each user must be established, resulting in a tree rooted at the server as seen previously in Fig. 1 and Fig. 2. For the rest of this paper the term *parent* and *child* will refers to the parent and child nodes in such a tree.

The main system components in this architecture are the video servers, the STBs, and the internal nodes (which include the local servers as well as the intermediate routers). The architecture described here is similar to a typical VoD architecture [11]. However, the main difference is that we require the internal nodes to possess storage capability and processing power.

The server's basic task is to fetch requested clips and send them over the network at the required rate for the display. The role of the STB is to request clips and perform playbacks. It is assumed that each STB has a small buffer to smooth out the incoming stream in order to minimize the jitter due to network latency.

It is at the internal nodes where the scheduled-multicast protocol (SMP) is employed. The next section describes in details the operation of this protocol and the signaling it offers to enable the interactions between the system components. The service model currently does not support VCR functions and we assume a uniform constant bit rate requirement for video transmission.

3.2 The Scheduled-Multicast Protocol (SMP)

The main goal of SMP is to facilitate the setup and tear down of SMP sessions, and the scheduling of the transmissions. An SMP session consists of the following:

- A buffer of a predefined size.
- A parent (the peer session which is sending the data),
- A list of children (peer sessions receiving data from this SMP session) with their respective buffer offsets.

The basic operations are described in Fig. 5. They are performed on a round basis and the time between rounds, referred to as the *round period*, is determined by: a) the message size being used and b) the bit rate requirement for the stream. We examine these factors in Section 4.2.

It is important to note that SMP does not specify the resource scheduling algorithm being employed. Instead, that task is left to a *scheduling layer* which determines the topology for the transmissions, and the buffer space required for each SMP session. Fig. 6 is a protocol graph which shows the layering of the protocols in our system.

SMP layer sits above the network transport layer so that it can make use of the services offered by the lower layers without having to worry about issues such as fragmentation and reassembly of messages. Furthermore, SMP can use a different network transport protocol with minimal changes.

SMP is separated from the resource scheduling so that it would be easy to install different scheduling algorithms without the need to modify the underlying operations of SMP. This gives rise to two sets of signaling messages: one for the interaction between the scheduling layer and the other for the peer-to-peer interactions of SMP. In SMP, the servicing of signals from the upper scheduling layer takes place between service-rounds while servicing peer-to-peer

Function: SMP Main Routine

- Loop forever:
  - Service a signal from the scheduling layer if required.
  - Call *SMP Service-Round*.
  - Sleep for the remaining time in the *round period*

Function: SMP Service-Round

- For each SMP session:
  - If the parent session has not finished sending the data:
    - Read any data packets from the parent session which have arrived since the last service-round and insert them into the current session buffer.
    - Read parent control information to see if it is the end of the data stream. If it is the end, mark this session as *finishing*.
  - Call *Service Children* with current session as the argument.
  - If the current session is *finished* then delete the session.

Function: Service Children (*smp_session*)

- For each child of *smp_session*:
  - Determine the child's offset in the buffer.
  - If offset is at the end of the buffer (ie. no more data in the buffer to send) **and** *smp_session* is marked as *finishing* then send control information to the child to indicate that it has finished sending data, and delete the child.
  - If offset is not at the end of the buffer, send to the child the data packet at that offset.
- If *smp_session* has no children and is marked as *finishing* then mark it as *finished*.

Figure 5. The basic operations of SMP

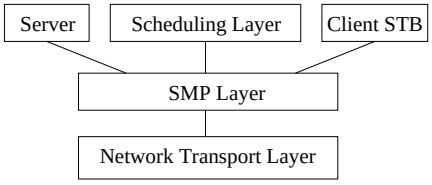

Figure 6. Our system protocol graph

signals is part of the service-round. Fig. 7 lists the message formats for the main signals, and the responses.

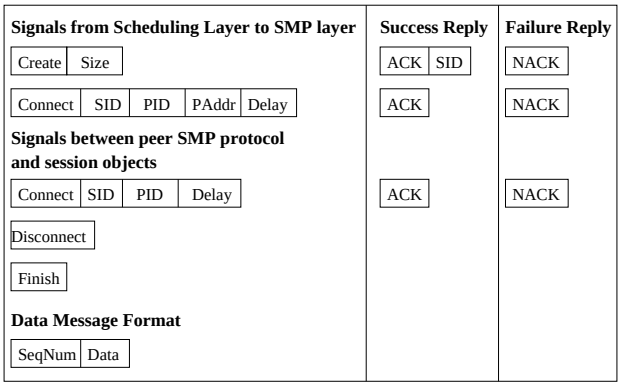

Figure 7. Signal and data message formats

The semantics of each message are described below:

- *Create* - The scheduling layer indicates to SMP to create a new session with buffer of size *Size*. If successful, a new session is created with a unique *SID*. SMP references it via a hash-table mapping *SID* to the session. The *SID* is returned to the scheduling layer.

- *Connect* - Can be sent from either the scheduling layer to SMP or between peer SMP sessions. In the first case, the scheduling layer is telling SMP session *SID* to connect to the parent session identified *PID* and its host address *PAddr*. The *Delay* is the offset required in the parent's buffer. After receiving this message from the scheduler, SMP strips the *PAddr* field to determine the parent host address. It then sends the same *Connect* message to the parent session. The *SID* is used by the parent session to identify the child session. If this succeeds a direct link between the child and the parent sessions would have been set up.

- *Finish* - Control message sent from the parent session to the child session to indicate the end of the stream.

- *Disconnect* - Sent from the child to the parent session to indicate that it no longer wishes to receive data.

- *Data* - Data messages which are sent between SMP sessions. The *SeqNum* field is a number which indicates the position of this message with respect to the video stream.

To illustrate how the signaling is used, we will use a sample scenario. Requests for a particular movie session are collected at the scheduling layer which uses that information to determine an optimum schedule. The schedule specifies the nodes where SMP sessions are needed (ie. the topology for the transmission tree), and how much buffer space each session requires.

The scheduling layer uses the CREATE signal to create the required sessions according to the schedule. The creation time for the sessions will be different depending on when each session is supposed to start receiving data, but a parent session will always be created before its child sessions. The scheduling layer follows the CREATE signal with a CONNECT signal specifying the ID of the parent session and the required delay. The actual connections are made by the SMP sessions sending CONNECT signals to their parents.

At each node where there is an active session, SMP performs its service-rounds reading data from the parent session and forwarding to its children the data at the required offset. When the server has finished transmitting the required clip, it sends a FINISH signal to all of its children. Each child session then goes into the *finishing* state sending a FINISH signal to each of its children after they have been sent all the data in the buffer destined for them. This continues down till it reaches the leaves of the tree, at which point all clients should have received their requested clip.

4 Implementation and Analysis

4.1 Implementation Details

To illustrate the working of SMP, we have built a prototype using the TCP/IP protocol stack and the Linux system socket interface. The Linux kernel was also modified using a technique developed by Hill et al. [7] to allow for finer time resolution (from 10ms down to $50\mu sec$)

SMP makes use of separate communication channels for the transfer of control information (such as the FINISH and DISCONNECT messages) and data. Each peer-to-peer session connection involves a TCP channel for the exchange of control information and a UDP channel for the transfer of data. TCP is used for signaling because it has the advantage of reliable delivery. This ensures that signals are never received out of order and it is possible to determine whether or not a particular signal has been successfully received. UDP was chosen as the protocol for data transport due to the following reasons:

- There is a time constraint on the delivery of multimedia data. If a packet cannot be delivered before its deadline, then there is no need to retransmit it.

- Ensuring that messages are sent in order has an impact on the playback since subsequent messages have to be delayed until the messages before them have been acknowledged. More often it is desirable to drop a message if later messages can be received in time and the

playback engine can employ various techniques available to compensate for that loss.

With regards to the implementation of the scheduling layer, there are two options: as a centralized service or as a distributed protocol with a complex set of signals for the negotiation of resource reservation and the associated routing. Since we are focusing on the design and implementation of SMP, we have decided to use the centralized approach.

A key aspect in implementing SMP is the data structure for the buffer. A naive implementation would simply allocate one block of memory for the buffer and then use it as a circular array. However, this will create problems when it comes to operations such as "resizing" or "splitting up" the buffer which are anticipated for the dynamic allocation of buffer space. At worst, this can involve copying large chunks of memory from one area to another leading to unacceptably long delays between service-rounds.

To avoid the above problem, we adopt a paging scheme similar to that found in common operating systems. Instead of allocating a large block of memory, the buffer is allocated many small fixed size blocks (referred to as pages) which it references through an array. In this manner, the memory can still be manipulated as a circular array, but to resize a large buffer, one can simply reorder the references to reduce or increase the number of pages referenced by the array with very little copying of the actual data. Similarly performing a "split" can be done efficiently by copying the page references rather than the pages themselves.

4.2 System Analysis

In this section, we examine the system and state the conditions which must hold for the timeliness delivery of data. Our analysis is based on a dedicated system. We also assumes that when a connection is setup there is sufficient bandwidth for the transmission.

For a given clip with bit rate R, and fixed message size, M_{size}, the message period T_{period} (the time difference between sending one message and the next) is:

$$T_{period} = M_{size}/R \tag{1}$$

Let T_{round} represents the time each service-round takes to run. Then to prevent congestion it must hold:

$$T_{round} \leq T_{period} \tag{2}$$

Thus for a 1.5Mb/sec bit rate requirement and a message size of 8KB, T_{round} must be less than 44ms. Hence the resolution of 10ms for the old Linux kernel is too large.

At this point, we make the assumption that the time it takes to service an incoming stream is equivalent to that of servicing a child. For an incoming stream, processing is needed to receive the data from the lower network, copy it into the socket buffer, determine where to insert it into the SMP-session buffer and finally copy it from the socket buffer to the SMP-session buffer. On the other hand to service the child, processing is required to determine the offset in the SMP-session buffer to retrieve the data from, to copy it into the socket buffer, and finally copy it from the socket buffer into the physical network. The two services have similar overheads and the only difference is that one copies from the socket buffer to the network while the other copies from the network to the socket buffer. The difference here is negligible so the assumption we make is quite valid. We denote the overhead of working out the offsets to be $T_{overhead}$ and the time to perform the copy from network to the buffer socket and later from the socket to the session buffer to be T_{msg}. If there are m sessions and n children belonging to the sessions, T_{round} can be expressed as:

$$T_{round} = (m+n) * (T_{overhead} + T_{msg}) \tag{3}$$

By using Equations 1 and 3, we rewrite Inequality 2 as:

$$M_{size}/R > (m+n) * (T_{overhead} + T_{msg}) \tag{4}$$

It is important to note that T_{msg} is not a constant, but depends on M_{size} itself. A large M_{size} leads to a large T_{msg} since it would take longer to copy the message from the network. The problem then lies in deciding what is the optimum message size such that the largest number of clients can be supported while Inequality 4 still holds. A large M_{size} will give us a longer T_{period} but the disadvantage is that it can introduce latency due to the increase in T_{msg}, thus overloading the underlying transport layer by writing a large amount of data within a short period of time. On the other hand, a small M_{size} results in a smaller period. For sessions with a large number of children, $T_{overhead}$ and T_{msg} will then be too large to satisfy Inequality 4. We will try to determine the optimum message size experimentally.

Another aspect of the system which needs to be looked at is whether the extra processing at each intermediate router will result in a greater variance in the packet arrival times. Furthermore, it is necessary to show that the above variance will not be magnified as the number of hops in the transmission path grows. This can lead to prolonged periods where there are too many packets arriving followed with too few, leading to buffer overflow or underflow at the client STB. However, an examination of the operations within a service-round shows that employing a session buffer at each intermediate node has quite the opposite effect. It smoothes out the stream delivery rate. This is because during each service round at most one message will be sent to each child session despite the fact that it may receive more than one message or no message during that round. To handle small bursts of data, the size of the SMP session buffers is set to be slightly larger than that specified by the scheduling layer

at creation time. Large bursts of data, which overflow the buffer, will result in packets being dropped. However, it is expected that there is sufficient bandwidth for all the transmissions, and therefore there should not be long periods of delays or bursts of messages.

5 Experimental Results

In our experiments, unless otherwise specified, the topology used is a simple three-level tree as shown in Fig. 1 with the clients connected to the intermediate router via a 100Mbit Ethernet LAN. From the intermediate router to the server is a 10Mb Ethernet link. The clip we use for testing has a bit rate requirement of 4Mbit/sec which is comparable with the bit rate requirement of a high quality MPEG 2 stream. The router is a Pentium Celeron 500 processor with 192MB of memory. In all the experiments involving SMP, two SMP sessions running on the router are used to service all the clients. This is chosen based on the fact that the network link from the server to the intermediate router is limited to 10Mbit/sec and therefore can only handle a maximum of two concurrent 4Mbit/sec streams.

To determine the optimal message size, the percentage of messages successfully received for varying numbers of concurrent clients is measured. Fig. 8 shows that for a small message size of 512 bytes, the overhead becomes significant as the number of concurrent clients grows. In this situation, we have $T_{round} > T_{period}$, leading to a large percentage of messages being dropped. A similar result is also evident for message sizes greater than 8192 bytes. However, the reason in this case is due to the overloading of the socket send-buffers when writing large-size messages to the network in a short period of time. The optimal message size shown in Fig. 8 ranges from 1024 to 4096 bytes. Messages in this range exhibit no message loss for up to 24 concurrent clients, and, moreover, they also exhibit identical percentage of message losses when the 100Mbit/sec bandwidth is exceeded at 25 or more concurrent clients. As a comparison, Fig. 8 also shows the results when SMP is not employed. Due to the 10Mbit bandwidth limitation from the server to the intermediate router, only two concurrent clients can be serviced without messages being dropped.

Another measure to determine the optimal message size is the CPU utilization of SMP at the router. This can be determined from the ratio T_{round}/T_{period}. Low CPU utilization means that the system is less susceptible to delays due to CPU contention. Fig. 9 shows the CPU utilization of SMP for increasing number of clients and for varying message sizes. As expected, the CPU utilization for small message sizes is very high due to the large overhead of processing more messages. However, for message sizes of 4096 bytes or greater, the CPU utilization is quite low at around 10%. Thus, from the data plotted in Fig. 8 and Fig. 9, we conclude that the optimal message size is 4096 bytes.

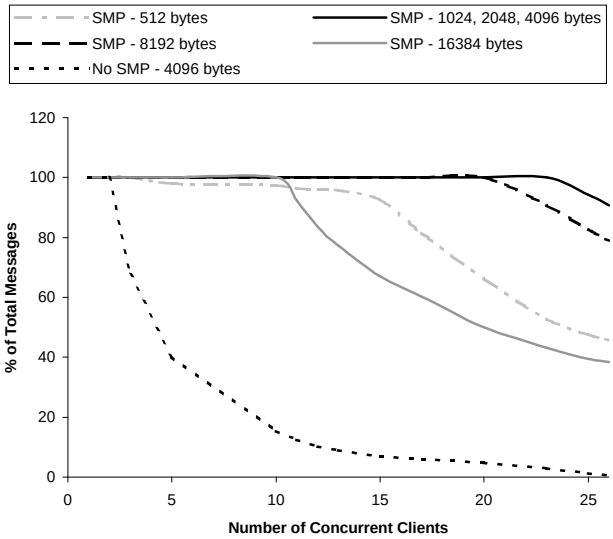

Figure 8. Percentage of messages received for increasing number of concurrent clients

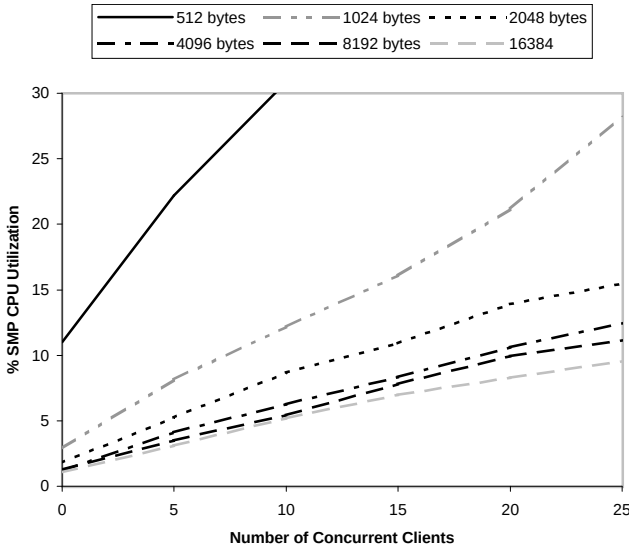

Figure 9. CPU utilization for increasing number of concurrent clients

At the client end, it is important to measure the rate at which data is received to determine whether or not SMP introduces additional delays that can cause a client's buffer to overflow or underflow, leading to *jittery* presentation of the clip. This can be measured by having the client reporting how much data it receives for each second duration. Fig. 10 is a portion of the trace which shows the average amount of

data received per second for a client. There are three traces: one for a system not employing SMP serving 2 concurrent clients, one using SMP servicing 24 concurrent clients, and one with 5 intermediate SMP sessions in the transmission path (instead of just one as in the simple three-level tree topology), again servicing 24 clients. The message size of 4096 bytes is chosen for all tests and during every one sec interval the server is sending out 121 messages (± 1 depending on the accuracy of the system timer) giving an expected data rate of 495000 bytes/sec.

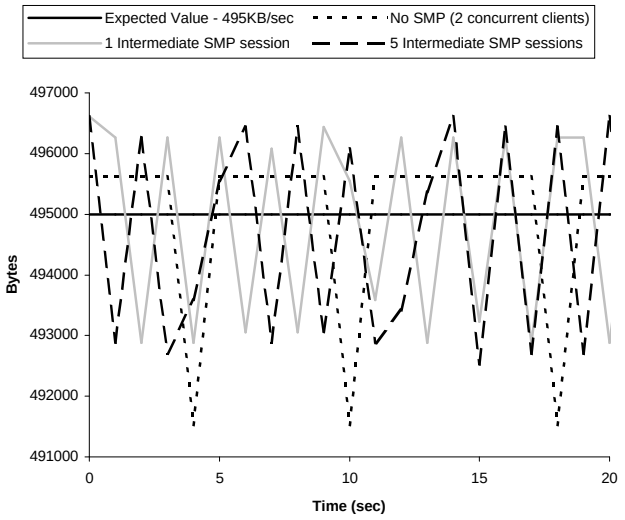

Figure 10. Average data rate received per client.

Fig. 10 shows that SMP introduces small variations in the incoming data rate compared with a system not employing SMP. However, this variation is very small, consisting of less than 1% of the expected received data rate. Further more, despite serving many more concurrent clients, and having more intermediate SMP sessions in the transmission path, this does not lead to long period of delays or long bursts of data. This is indicated by the fact that each peak in the trace is followed by a trough and vice versa. Thus employing a small buffer at the client will effectively smooth out the stream removing the observed peaks and troughs.

6 Future Work

Our future work concentrates on improving SMP and providing algorithmic support to it. In the first instance, we are looking at using ATM as the underlying transmission protocol which has the advantages of offering guarantees on different"Quality of Services". For the latter, scheduling decisions made at the scheduling layer requires designing algorithms for optimal resource utilization. We will also extend the scope to *on-line* algorithms since our goal is to develop a VoD system that is capable of handling *dynamic* situations where requests are not known in advance.

References

[1] C. Aggarwal, J. Wolf, and P. Yu. On Optimal Batching Policies for Video-on-Demand Servers. In *IEEE Multimedia Computing and Systems Conference*, pages 253–258, 1996.

[2] C. Aggarwal, J. Wolf, and P. S. Yu. On Optimal Piggyback Merging Policies for Video-On-Demand Systems. In *Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '96)*, pages 200–209, 1996.

[3] P. Basy, A. Narayanan, R. Krishnan, and T. Little. An Implementation of Dynamic Service Aggregation for Interactive Video Delivery. In *Proc. SPIE – Multimedia Computing and Networking*, January 1998.

[4] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with batching. In *ACM Multimedia Conference*, pages 15–23, 1994.

[5] M. Garofalakis, B. Özden, and A. Silberschatz. On periodic resource scheduling for continuous media databases. *The VLDB Journal 7*, 4:206–225, 1998.

[6] L. Golubchik, J. C. Lu, and R. Muntz. Reducing I/0 Demand in Video-On-Demand Storage Servers. In *Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '95)*, pages 25–36, 1995.

[7] R. Hill, Balaji, S. Pather, and D. Niehaus. Temporal Resolution and Real-Time Extension to Linux. Technical Report ITTC-FY98-TR-11510-03, University of Kansas, June 1998.

[8] H. W. Holbrook and D. R. Cheriton. IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications. In *Proceedings of ACM SIGCOMM'99*, pages 65–78, 1999.

[9] M. Kamath, K. Ramamritham, and D. Towsley. Continuous Media Sharing in Multimedia Database Systems. In *Proceedings of the Fourth International Conference on Database Systems for Advanced Applications (DASFAA '95)*, pages 79–86, 1995.

[10] R. Krishnan and T. D. C. Little. Service Aggregation Through Rate Adaptation Using a Single Storage Format. In *Proc. 7th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, May 1997.

[11] T. D. C. Little and D. Venkatesh. Propects for interactive video-on-demand. *IEEE Multimedia*, 1(3):14–24, 1994.

[12] S. McCanne. Scalable Multimedia Communication: Using IP Multicast and Lightweight Sessions. *IEEE Internet Computing*, 3(2):33–45, March/April 1999.

[13] B. Özden, R. Rastogi, and A. Silberschatz. On the Storage and Retrieval of Continuous Media Data. In *3rd International Conference on Knowledge Management*, pages 322–328, 1994.

[14] D. A. Patterson, G. A. Gibson, and R. H. Katz. A case for Redundant Arrays of Inexpensive Disks(RAID). In *Proceedings of the Conference on Management of Data*, pages 109–116, 1988.

[15] W. Shi and S. Ghandeharizadeh. Buffer Sharing in Video-On-Demand Servers. In *Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '97)*, pages 13–20, 1997.