A SMOOTHING PROXY SERVICE FOR VARIABLE-BIT-RATE STREAMING VIDEO

Jennifer Rexford
Networking & Distributed Systems
AT&T Labs – Research
Florham Park, NJ 07932
jrex@research.att.com

Subhabrata Sen
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
sen@cs.umass.edu

Andrea Basso
Speech & Image Processing
AT&T Labs – Research
Red Bank, NJ 07701
basso@research.att.com

Abstract— **Provisioning network resources for multimedia streaming is complicated by the bursty, high-bandwidth traffic introduced by compressed video, as well as the variability of the throughput, delay, and loss properties of the Internet, and the lack of end-to-end control by any one service provider. To address these problems, we propose that proxies should perform online smoothing by transmitting frames into the client playback buffer in advance of each burst, to reduce network resource requirements without degradation in video quality. This paper describes the practical systems issues we have encountered in building a smoothing proxy service using off-the-shelf components, in the context of an MPEG-2/RTP streaming testbed.**

I. Introduction

Distributed audio and video applications, such as distance learning and entertainment services, rely on the efficient transmission of multimedia streams. Compressed multimedia streams exhibit significant burstiness at multiple timescales, due to variations in detail and motion within and between scenes, as well as the frame structure of the encoding scheme [1, 2]. The encoder could avoid generating a bursty stream by adjusting the quantization level of the frames across time, at the expense of introducing fluctuations in audio/video quality. However, for the same average bandwidth, a variable-bit-rate encoding offers higher quality and more opportunities for statistical multiplexing gain than would be possible for a constant-bit-rate encoding [2, 3]. Furthermore, burstiness is inherent in emerging orchestrated media applications that combine video, audio, and text sources in a single stream.

Despite the benefits of variable-bit-rate *encoding*, the variability of frame sizes complicates the *transmission* of multimedia streams. End-to-end mechanisms for handling bursty traffic are difficult to deploy, since a single network service provider typically does not control the entire path from the source to the end client(s). Furthermore, the best-effort service model of the Internet introduces wide variation in the network delay, throughput, and loss properties. We contend that service providers can address these problems by deploying *proxy services* that improve transport efficiency and video quality. In particular, we propose that proxies should perform *online smoothing* of variable-bit-rate video to reduce the network resource requirements for transmission to the client(s). For example, smoothing could occur at network access or peering points. We focus on providing high-quality, non-interactive video to the growing number of users with broadband access.

For a wide range of videocast applications, such as live sporting events or prerecorded movies, many users may be willing to tolerate a playback delay of several seconds or even minutes in exchange for higher quality and/or a lower price. These applications differ from services for streaming live, interactive video, and from traditional video-on-demand transmission of prerecorded content from a server. Due to tight delay constraints, smoothing of live, interactive video targets *small* timescale burstiness (say, within an MPEG group-of-pictures) and relies on reducing the encoding quality during scenes with high bandwidth requirements. In contrast, smoothing of a stored, prerecorded stream can target longer timescale burstiness by transmitting frames into the client playback buffer in advance of each burst, *without degradation in the video quality* [4]. Compressed, constant-quality video streams often have substantial medium-term burstiness [5].

Ideally, the originating server would perform the smoothing function, particularly for prerecorded streams where the server could compute the transmission schedule in advance. However, the content provider may not provide a smoothing service, and may lack sufficient information about the client's buffer size, delay tolerance, and playback capabilities. Instead, a network provider could offer smoothing services at a proxy. Smoothing inside the network introduces new challenges, since the proxy does not store the entire video in advance. This limits the proxy's latitude in transmitting frames ahead of the client playback time, and requires an online computation of the transmission schedule without knowledge of future frame sizes. Similar issues would arise at an origin server that performs online smoothing of a live video stream.

In our earlier work [6, 7], we have found that these practical constraints do not significantly reduce the benefits of workahead smoothing, as discussed in Section 2. In fact, introducing a 1–60 second smoothing delay can substantially reduce the variability of the bandwidth requirements for high-quality video; for prerecorded video, it is possible to hide this start-up delay by integrating online smoothing with effective caching techniques [8, 9]. Reducing the burstiness of individual streams is particularly important for supporting high-bandwidth multimedia applications on broadband access media, which provide limited opportunities for multiplexing gain. For example, a high-quality MPEG-2 stream consumes 4–6 Mbps, and emerging hybrid fiber-coax networks typically provide a 27 Mbps downstream channel that is shared between several users.

Although proxies have been proposed for a number of multimedia services, such as transcoding and retransmission [10, 11], online smoothing introduces new challenges in controlling the low-level *timing* of frame and packet transmissions. Addressing these systems challenges in a commercial off-the-shelf platform is the focus of this paper. Section 3 extends our online smoothing model to accommodate practical constraints on observing packet boundaries, controlling packet scheduling, and computing the transmission schedule. Through experiments using a 10-minute MPEG-2 video clip, we show that it is possible to address these issues without sacrificing the effectiveness of online smoothing. Based on these results, we are building a prototype of the proxy smoothing service in a PC-based MPEG-2/RTP video streaming testbed [12]. Section 4 discusses our implementation experiences in Windows NT, and describes how we coordinate packet reception, schedule computation, and packet transmission while minimizing data copying. Section 5 concludes the paper with a summary of future research directions.

II. Online Smoothing

Bandwidth smoothing has the potential to substantially reduce the variability of network resource requirements for transmitting constant-quality video. To motivate the proxy architecture, we briefly summarize our previous work on online smoothing.

Smoothing model: The basic architecture of the proxy smoothing service draws on earlier work on online smoothing of variable-bit-rate video [6, 7]. We assume a discrete-time model, where one time unit corresponds to the time between the playback of successive frames; for a 30-frames/second full motion video, a *frame time* corresponds to $1/30^{th}$ of a second. The proxy has a B_P-bit buffer and receives A_i bits of the stream by time i, as shown in Figure 1. The value of A_i is not known in advance, and is dictated by the frame sizes and the jitter in the path between the server and the proxy. The client initiates playback at time $w > 0$, allowing time for the proxy to perform workahead transmission into the client playback buffer. The client has a B_C-bit buffer and must receive at least D_{i-w} bits of the stream by time i to permit continuous decoding and playback. For an MPEG stream, the value of D_i includes any data that must arrive before decoding frame i (e.g., a P frame necessary to encode a B frame in a $IBBPBBP$... sequence). In the absence of jitter or smoothing at upstream nodes, the proxy arrival vector is simply a shifted version of the client's playout vector (i.e., $A_i = D_{i+w}$).

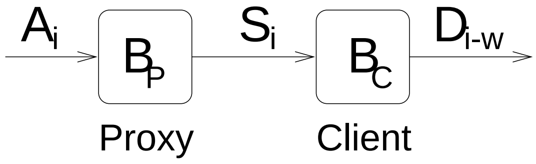


Fig. 1. Online smoothing model

Smoothing constraints: The proxy computes a schedule S_i for the number of bits to transmit by time i, based on constraints that avoid underflow and overflow of the two buffers. For the client buffer, this requires the proxy to transmit at least D_{i-w} and at most $D_{i-w} + B_C$ by time i. Similarly, to avoid underflow and overflow of the proxy buffer, the proxy must transmit at most A_i and at least $A_i - B_P$ by time i . Hence, a *feasible* transmission schedule must satisfy

$$\max\{D_{i-w}, A_i - B_P\} \leq S_i \leq \min\{D_{i-w} + B_C, A_i\}$$

for $i = 0, 1, \ldots, N + w$ for a video with N frames and a smoothing window of w frame times.

Transmission schedule: In general, all the future frame sizes in the video may not be known in advance at the proxy, since the video is streaming by. The proxy therefore *incrementally* computes the constraint curves online, as more frames arrive. The w frame time delay between the arrival and client consumption curves allows the proxy to know at time i, D_j for some window of time $j = i + 1, i + 2, \ldots$ into the future. This allows the proxy to compute, at time i, an estimate of the lower and upper constraints for the time window, as shown in Figure 2. Based on the computed lower and upper constraints, the proxy then computes the smoothed schedule S for the same time window. Starting with the current position of the transmission schedule S_i at time i, the algorithm sequences across

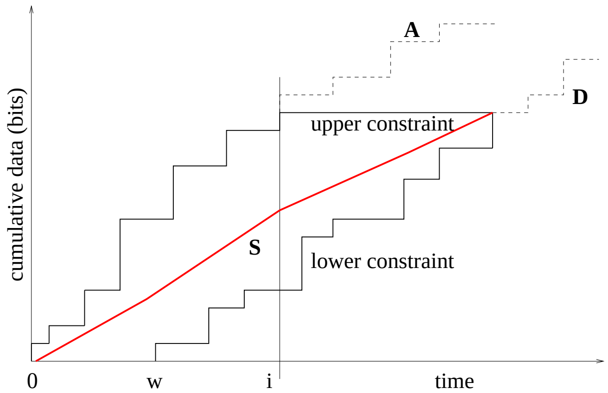


Fig. 2. Online smoothing constraints

the time slots $j = i+1, i+2, \ldots$, iteratively computing the transmission rates for times $i+1, i+2\ldots$. The transmission rate for time k is the slope of the S curve (i.e., $S_k - S_{k-1}$).

Schedule computation: The proxy incrementally computes the schedule S in this online, window-based manner, using algorithms initially developed for smoothing prerecorded video. These algorithms achieve their computational efficiency by finding a trajectory that stays between the constraint curves, without imposing any restrictions on transmission rates or packet boundaries. For our online algorithm, we adapt the $O(N)$ shortest-path algorithm [13] that precomputes a schedule S for streaming prerecorded video with the minimum peak and standard deviation of the transmission rates. Executing online smoothing in every time slot would allow the proxy to incorporate the most recent arrival information in the schedule computation, at the expense of increased processing overhead. Instead, the proxy computes a schedule every α time units, where α is a tunable parameter between 1 and w. This is particularly important in scaling the proxy to smooth multiple simultaneous video streams. Relatively simple frame-size prediction schemes can also be used to estimate the values of D_j and A_j further into the future, to allow a more aggressive computation of the schedule S_j [7].

Performance properties: Applying the online smoothing model to a collection of MPEG-1 and motion-JPEG video traces illustrates that a smoothing window (w) of 1–60 seconds and using 1–8 megabytes of storage at the proxy and the clients (B_P and B_C) offers substantial reductions in the peak and variability of the transmission rates [7]. Smoothing at this timescale enables the proxy to remove burstiness beyond the inherent variability of the MPEG group-of-pictures structure. By smoothing over scenes with different average frame sizes, our online algorithm reduces the peak bandwidth by an *additional* factor of two, depending on the video[1]. With the decreasing cost of high-speed memory, video set-top boxes and personal computers can easily devote several megabytes of buffer space to video smoothing, in exchange for these substantial reductions in network resource requirements. We also find that computing the schedule at relatively coarse time intervals (e.g., $\alpha = w/2$ or $\alpha = w/3$) is sufficient to achieve nearly all of the gains of online smoothing. More detailed performance results are presented in [7].

III. Practical Constraints

The promising results in Section 2 suggest that it is possible to build a proxy smoothing service from off-the-shelf components. However, a real implementation must deal with deviations from the higher-level abstractions in the smoothing model, as well as system overheads introduced by the underlying platform (particularly when the proxy handles multiple video streams simultaneously). This section describes how the smoothing model can be generalized to accommodate practical constraints without sacrificing the key properties of the more abstract model.

A. Packetized Transmission Schedule

Despite the value of the abstractions in Section 2, the online smoothing model includes a number of simplifying assumptions that do not hold in a real system. In particular, the smoothing algorithm can generate transmission schedules S that send data continuously at arbitrary rates. However, in reality, each video frame consists of some number of packets (e.g., 15 or 30), and the proxy must deal with the video stream as the sequence of packets generated by the server. The packet size cannot be very small, as such fine-grain packetization would introduce substantial overheads for packet header information. Hence, the proxy cannot obey the computed transmission schedule S at an arbitrarily small timescale. Still, the computation of the fluid schedule S provides a very simple and effective way to compute the proper spacing between successive

[1] The optimal offline algorithm could conceivably achieve much better results, by targeting long-term burstiness and by knowing all of the frame sizes in advance. Yet, our experiments show that these effects are not very significant. Most of the burstiness in the underlying streams stems from the short-term and medium-term variations in frame sizes within and between scenes. Removing significant long-term burstiness, when present, requires a very large client buffer and high start-up latency, even for offline smoothing.

packets. In particular, a packet can be transmitted at the time when its first bit is scheduled in S. This enables the proxy to obey the transmission schedule at the timescale of individual packets. In effect, S is converted from a piecewise linear curve into a staircase function, where packets are transmitted at the link rate, with an inter-packet spacing that is dictated by the desired transmission rate.

B. Packet Scheduling Granularity

Ideally, the spacing of individual packets would be performed by a hardware adapter that performs traffic shaping. However, existing shaping devices for IP networks do not provide such dynamic, fine-grain control over the transmission rates for each stream. We therefore assume that the actual packet scheduling must be done by the proxy software itself.

System overheads may make it impossible for the proxy to schedule transmissions for multiple video streams at the level of individual packets. For example, a video transmission at 30 frames/second with 15 packets/frame requires an *average* inter-packet spacing of only 2.2 msec. We therefore assume that the proxy transmits a small set of packets in a single step; we define the *transmission granularity* T as the smallest timescale at which the proxy can transmit the video. In particular, the proxy initiates transmission of any packets that would be scheduled within the next T milliseconds according to the smooth transmission schedule; these packets may span one or more rate changes in the schedule. In the worst case, the proxy's communication subsystem would transmit these packets back-to-back at the link rate, resulting in additional burstiness in the traffic at the very small time scale. Fortunately, such small time-scale variability can easily be absorbed by the network components between the proxy and the client. These restrictions in coordinating the fine-grain packet spacing do not limit the online algorithm's ability to remove the medium timescale variability.

As T increases, more packets will be transmitted back-to-back at any time. More packets will therefore arrive at the client buffer earlier than dictated by the smoothed schedule. For relatively small transmission granularities ($T \leq 1$ frame time), the resulting transmission schedule still obeys the upper and lower constraint curves, since these constraints change only at frame boundaries. But, for larger values of T, the resulting schedule may send aggressively enough to overflow the client buffer. We highlight this effect in Figure 3(a), which plots the client buffer utilization across time for a ten-minute MPEG-2 clip of *Space Jam*, variable-bit-rate encoded with a peak rate of 12 Mbps and a quantizer setting of 8, using the platform described in Section 1. The clip has an average bandwidth of 3.1 Mbps, and the client buffer size used for computing the smoothed fluid schedule is 512 KB. The graphs in Figure 3(b) and Figure 3(c) show that even for a large 30-second smoothing window, for a range of reasonable T values, the maximum excess buffer requirement is only a few hundred KB and excess buffering is required very rarely. This suggests that the occasional overflows of the client buffer can be easily avoided by a small amount of overprovisioning at both the proxy and the client; such overprovisioning is likely to be necessary anyway to handle jitter introduced by the network. Another alternative would be an elastic playback buffer allowing for occasional "overflow" at the client, where the excess buffer beyond B_C is time-multiplexed with other client applications.

C. Schedule Computation

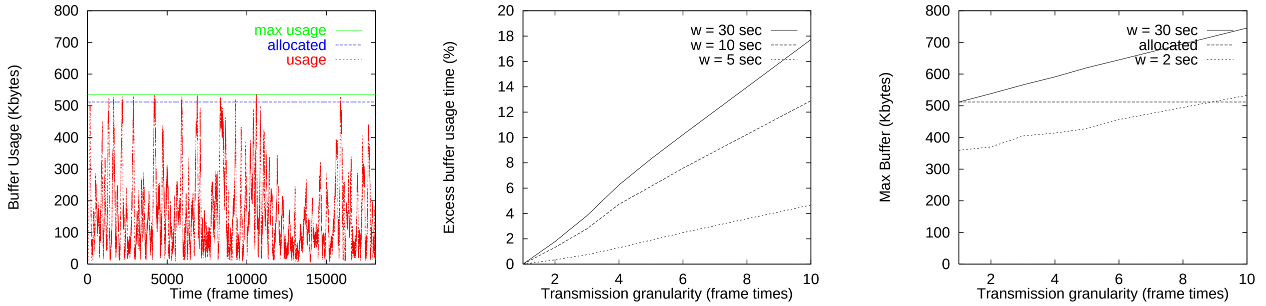
The abstract online smoothing model presumes that the proxy can compute the transmission schedule S in real-time, as frames arrive. As discussed in Section 2, the computational complexity is linear in the size of the smoothing window. We measure the processing overheads of the algorithm on a lightly-loaded 333 MHz Windows NT PC, with the smoothing code running at real-time priority. The experiment varies the window size w, with $\alpha = w/2$ and $B_C = 1$ MB, and reports the maximum and average processing time, over the 10-minute clip of *Space Jam* in Figure 4. Each value is averaged over six independent computations across the entire clip. The graph shows that the processing time increases linearly in w, and stays within 1.2 msec for windows less than 30 seconds long. These results suggest that an off-the-shelf commodity system configuration could easily offer smoothing services for a few tens of video streams.

IV. Prototype Proxy Service

Based on the results in Section 2 and Section 3, we are building a prototype of the online smoothing proxy service on a PC running Windows NT. After a brief description of the video streaming platform, we describe the software architecture and key design decisions in the proxy implementation.

A. LiveNet MPEG-2 Platform

The smoothing proxy is being implemented and evaluated in the context of the LiveNet MPEG-2 multimedia platform [12, 14], as shown in Figure 5. For ease of



(a) Buffer utilization vs. time (b) Percent time with excess buffer (c) Maximum client buffer usage

Fig. 3. Utilization of client buffer

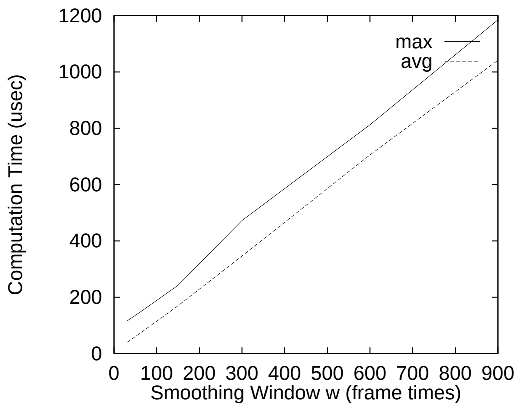


Fig. 4. Processing requirements for computing schedule S

debugging and performance tuning, the video source was initially implemented on two PCs that perform MPEG encoding and RTP packetization; a more recent version of the platform can perform both functions on a single high-end PC. The digitized output of a video source is fed to an MPEGXpress MPEG-2 audio/video encoder which generates a variable-bit-rate stream. The second PC parses the MPEG-2 stream, performs RTP packetization (following RFCs 1889, 1890, and 2055), and pacets the transmission of the RTP packets. The third PC executes the smoothing algorithm and shapes the output stream. The fourth PC is the client, which includes an RTP depacketizer, a network adaptation layer, and a hardware video/audio MPEG-2 decoder. Each machine runs a minimal amount of additional software (e.g., screen savers and daemons are disabled) and, on the Windows NT machines, the video streaming software runs at real-time priority.

B. Proxy Software Architecture

The smoothing software running on the proxy consists of three main modules for receiving, scheduling, and transmitting packets, as shown in Figure 6. Solid lines represent the flow of packet data from the incoming UDP socket to the outgoing UDP socket, whereas the dashed lines illustrate the flow of control to and from the module that executes the smoothing algorithm. Arriving RTP packets are stored in the proxy memory, and are associated with the appropriate frame based on the timestamp field in the RTP header; packets in the same frame have the same timestamp. Then, the size of the frame is updated to incorporate the new packet. This size information is used by the smoothing algorithm to compute the transmission schedule. Each invocation of the smoothing algorithm computes the transmission schedule for the next α time units, based on the most recent frame size information. Every $T \leq \alpha$ time units, the send module initiates transmission of packets that should be sent in this time interval. The appropriate number of packets is determined based on the transmission rates and the packet-size information.

C. Process Structure

The proxy must carefully coordinate the packet reception, scheduling, and transmission operations to achieve the potential gains of online smoothing. If the proxy falls behind in receiving incoming packets and accumulating frame sizes, then the smoothing algorithm does not have access to all of the frames, resulting in a suboptimal transmission schedule. In an extreme situation, delaying packet reception can introduce additional data copying, or even packet loss. On the other hand, if the packet reception module runs too frequently, the schedule-computation and send modules may not execute in a timely manner. If the smoothing algorithm cannot run often enough (large α), the resulting schedule may not be very smooth. In an extreme situation, delayed execution of the smoothing algorithm may cause starvation of the send module; to avoid this prob-

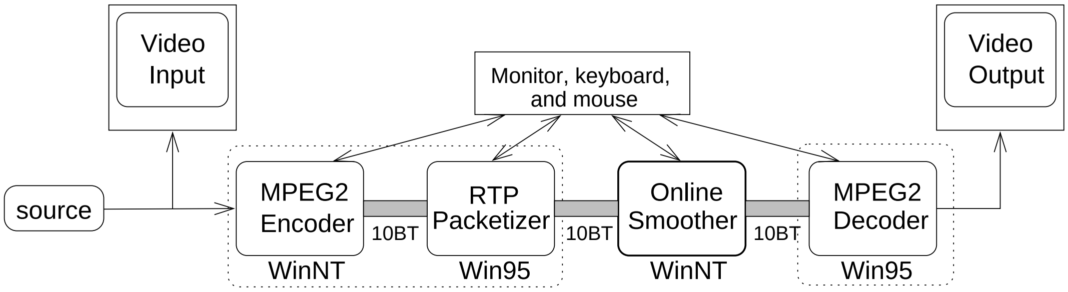


Fig. 5. MPEG-2 video streaming platform

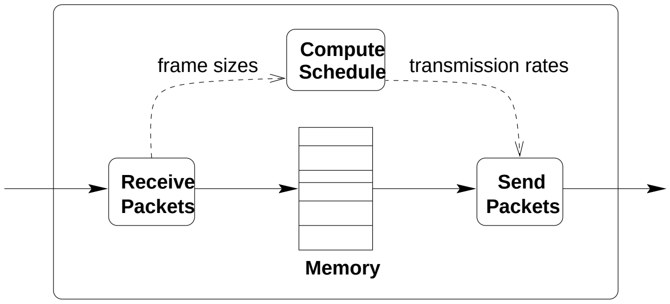


Fig. 6. Software architecture of proxy smoothing service

lem, the send routine could optimistically continue to transmit data at the last known rate, though this may undermine some of the benefits of smoothing. Similarly, neglecting to run the send routine in a timely fashion can introduce additional burstiness in the outgoing traffic. In the worst case, delay in executing the send routine could result in underflow of the client playback buffer.

The three modules could conceivably run as separate threads or processes. However, Windows NT does not provide fine-grain control over processor scheduling. In addition, context-switch overheads can be both large and highly variable, particularly in relation to the time granularity of packet arrivals and departures. Finally, executing the modules in separate processes would require semaphores to control access to shared data structures. In our implementation, the proxy executes the three modules in a single process, and coordinates the division of processing resources. Within the single-process architecture, each of the modules could conceivably be implemented as callback routines, invoked by a high-resolution timer. Although the Windows Win32 interface does provide a *multimedia timer* facility [15], the invocation of callback routines typically involves a large amount of jitter, making it difficult to control events below thresholds of tens of milliseconds [16].

Instead, the proxy schedules the three modules using the *high-performance counter* in Windows NT [15], effectively polling the clock to determine which module to invoke next. The invocation frequency of each module depends on the key parameters in the smoothing model. For example, the smoothing algorithm must run every α time units, and the send module must be invoked every T time units. The receive module should run as frequently as possible, subject to the constraints of the other two modules. Though it may be possible to run the smoothing algorithm to completion for small window sizes w, large windows may consume too much processing time, requiring occasional inspection of the counter to schedule the send and receive routines. Fortunately, the smoothing algorithm operates as a loop with many iterations, simplifying the scheduling of calls to the send and receive modules. We are in the process of fine-tuning the prototype implementation of the proxy, allowing us to experiment with the parameters in the smoothing model (e.g., w, α, and T), as well as the polling frequency for the high-performance counter.

D. Memory Management

The proxy must receive and send packets at a high rate, in the range of 4–6 Mbps *on average* for each high-quality MPEG-2 stream. Supporting multiple streams requires careful control of packet reception to ensure that packets do not overflow the kernel receive buffer, or the network adapter. Also, the proxy should not introduce extra data copying, particularly since the smoothing function does not modify the contents of the UDP packets. Data copying is minimized by exploiting the *asynchronous overlapped I/O* model in Windows NT [15]. Overlapped I/O allows the application to perform asynchronous packet reception and transmission by posting one or more buffers to the kernel. This enables the kernel to copy data directly from the network device into user space. The proxy uses non-blocking receive calls, and checks the status of the call upon the next invocation of the receive module. The send module initiates packet transmission directly from user space by passing a pointer to the appropriate buffer to the kernel. This avoids copying data from user to kernel space. Upon completion of the blocking send call, the send module changes the status of the buffer from busy to idle.

When no receive buffers are posted, an arriving packet must be copied from the network adapter into the kernel, followed by a second copy into user space when the receive module issues a new buffer. If the kernel receive buffer is full, the packet is discarded. To avoid unnecessary data copying and packet loss, the receive module maintains up to m outstanding buffers at any time. Larger values of m provide greater latitude in deciding when to invoke the receive module, allowing the proxy to devote more attention to the send module, to reduce the burstiness of the outgoing traffic. The receive module could conceivably allocate these m buffers from a single region of memory. However, the receive module does not know the packet size in advance, forcing the allocation of a large enough buffer to handle the maximum packet size for each receive event. With a variable-bit-rate encoding, the largest packet may be much bigger than the average packet. Since allocating from a single shared memory could result in substantial fragmentation, the packet memory is divided into multiple contiguous regions, each acting as a circular buffer for handling one outstanding receive event[2]. We are experimenting with our prototype implementation to determine the appropriate number of circular buffers to balance the trade-off between data copying and memory fragmentation.

V. Conclusion

Proxy services can play an important role in improving transport efficiency and user performance for streaming audio and video in the Internet. In this paper, we have described how to perform online smoothing of variable-bit-rate streams, and addressed many of the practical issues that arise in building such as service. Then, we discussed the key design decisions in our prototype implementation on Windows NT, as part of the LiveNet MPEG-2/RTP testbed. As we complete the prototype of the smoothing service, we plan to measure the burstiness properties of the outgoing stream as a function of the parameters that control packet reception, schedule computation, memory management, and packet transmission. We are also extending the online smoothing service to include caching of the initial frames of popular prerecorded video streams, to decouple the smoothing window size from the client start-up latency [8, 9]. By integrating *prefix caching* with online smoothing, the proxy can achieve all of the advantages of using a larger smoothing window, without increasing client playback delay.

[2]Occasional fragmentation can still occur within one of these circular buffers when RTP packets arrive out of order at the proxy. The send module reorders the out-of-order packets, which introduces temporary gaps in the circular buffers.

References

[1] M. Garrett and W. Willinger, "Analysis, modeling and generation of self-similar VBR video traffic," in *Proc. ACM SIGCOMM*, September 1994.

[2] T. V. Lakshman, A. Ortega, and A. R. Reibman, "Variable bit-rate (VBR) video: Tradeoffs and potentials," *Proceedings of the IEEE*, vol. 86, pp. 952–973, May 1998.

[3] I. Dalgic and F. A. Tobagi, "Performance evaluation of ATM networks carrying constant and variable bit-rate video traffic," *IEEE J. Selected Areas in Communications*, August 1997.

[4] W. Feng and J. Rexford, "Performance evaluation of smoothing algorithms for transmitting prerecorded variable-bit-rate video," *IEEE Trans. Multimedia*, pp. 302–313, September 1999.

[5] M. Krunz and S. K. Tripathi, "On the characteristics of VBR MPEG streams," in *Proc. ACM SIGMETRICS*, June 1997.

[6] J. Rexford, S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, and D. Towsley, "Online smoothing of live, variable-bit-rate video," in *Proc. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 249–257, May 1997.

[7] S. Sen, J. Rexford, J. Dey, J. Kurose, and D. Towsley, "Online smoothing of variable-bit-rate streaming video," Tech. Rep. 98-75, University of Massachusetts – Amherst, 1998.

[8] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE INFOCOM*, March 1999.

[9] S. Gruber, J. Rexford, and A. Basso, "Design considerations for an RTSP-based prefix-caching proxy for multimedia streams," Tech. Rep. 990907-01, AT&T Labs – Research, September 1999.

[10] E. Amir, S. McCanne, and H. Zhang, "An application level video gateway," in *Proc. ACM Multimedia*, November 1995.

[11] N. F. Maxemchuk, K. Padmanabhan, and S. Lo, "A cooperative packet recovery protocol for multicast video," in *Proc. International Conference on Network Protocols*, October 1997.

[12] A. Basso, G. L. Cash, and M. R. Civanlar, "Implementation of a real-time MPEG-2 video/audio delivery system based on RTP." In submission, December 1998.

[13] J. D. Salehi, Z.-L. Zhang, J. F. Kurose, and D. Towsley, "Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing," *IEEE/ACM Trans. Networking*, vol. 6, pp. 397–410, August 1998.

[14] A. Basso, G. L. Cash, and M. R. Civanlar, "Transmission of MPEG-2 streams over non-guaranteed quality-of-service networks," in *Proc. Picture Coding Symposium*, 1997.

[15] B. Guinn and D. Shute, *Windows Sockets Network Programming*. Addison-Wesley, November 1995.

[16] M. B. Jones and J. Regehr, "Issues in commodity operating systems for time-dependent tasks: Experiences from a study of Windows NT," in *Proc. Workshop on Network and Operating System Support for Digital Audio and Video*, July 1998.