Adaptive Live Video Streaming by Priority Drop

Jie Huang, Charles Krasic, Jonathan Walpole, and Wu-chi Feng
OGI School of Science and Engineering
Oregon Health and Science University
{jiehuang, krasic, walpole, wuchi}@cse.ogi.edu

Abstract

In this paper we explore the use of Priority-progress streaming (PPS) for video surveillance applications. PPS is an adaptive streaming technique for the delivery of continuous media over variable bit-rate channels. It is based on the simple idea of reordering media components within a time window into priority order before transmission. The main concern when using PPS for live video streaming is the time delay introduced by reordering. In this paper we describe how PPS can be extended to support live streaming and show that the delay inherent in the approach can be tuned to satisfy a wide range of latency constraints while supporting fine-grain adaptation.

1. Introduction

Scalable video surveillance systems, where potentially thousands of cameras are involved, will require the underlying networking and coding mechanism to be scalable, efficient, and adaptive. In particular, the video cameras in these systems in aggregate can easily overload the network that connects them. As a result, we expect that in such systems small computing resources will be placed with each camera that allows it to deal with the resource constraints. We believe that such resources should be used to help make the system as scalable as possible and to provide the highest quality video.

Priority-progress streaming (PPS) is such an adaptive streaming mechanism [5] [6]. It uses a time-window-based approach in which all data packets with timestamps within a certain period of time are placed in a window and reordered into priority order before transmission. It then transmits these packets for the time duration of the window only. At the end of the window duration, it discards unsent packets and moves on to the next window. In this way, the available bandwidth is used to send the most important elements of the stream and the least important elements are dropped. Our implementation of PPS in the Quasar video pipeline [5] shows that PPS is good for streaming stored video.

The reordering window in PPS introduces latency, however, and this latency might be problematic for video surveillance applications, which stream live video. In live video streaming, the latency characteristics of the streaming mechanisms partially determine the freshness of the video content. The freshness of the video content, which is measured by the end-to-end latency from a frame being captured to its display, tends to be important for video surveillance applications.

In this paper, we explore how much of a problem the latency in PPS is for live video streaming and determine the range of video surveillance applications it can support. We describe how PPS can be extended to support live video streaming, and evaluate the latency implications of the approach. Our implementation of the live Quasar pipeline shows that even with fairly rudimentary scalable video encoding technology, the latency due to adaptation in PPS can be reduced to as little as 400ms while maintaining fine-grain adaptation. This means that applications with a latency tolerance of a half second can be supported using TCP-friendly protocols on a coast to coast link in the US (where propagation delay is typically less than 100ms).

This paper is organized as follows. Related work is discussed in Section 2. Section 3 introduces the basic idea of PPS and describes how it works for stored video streaming. Section 4 discusses the problems of using PPS for live video streaming. Section 5 outlines a series of experiments for evaluating the latency and adaptation granularity characteristics of PPS and presents results. Finally, Section 6 concludes the paper and discusses future work.

2. Related work

Streaming video adaptively involves many research areas. Wu et al. have written a comprehensive survey of video streaming approaches and directions in their paper [11]. Vandalore et al. give a detailed survey of application level adaptation techniques [10].

A common approach to adaptive live streaming is to monitor network conditions using feedback-based

mechanisms such as RTCP receiver reports [1] and adjust video encoding parameters on the fly [4][8] so that the rate of the encoded video stream matches a dynamically determined target bandwidth. A key advantage of this approach is compression efficiency—the video encoder is able to optimize video quality for the given target bandwidth. Another advantage is its support for fine-grain adaptation—the target bandwidth can be chosen from a continuous range. A third advantage is low latency—adaptation can be performed without reordering data. The main disadvantages of the approach are its inability to satisfy conflicting requirements of heterogeneous receivers in a simulcast or multicast distribution network, the difficulty of tuning encoding parameters to achieve optimal video quality for a certain video rate, and the difficulty of tuning the feedback control to determine the suitable and accurate target video rate. If the target video rate is chosen incorrectly it will either result in network underutilization, congestion, or increased delay.

PPS takes an alternative approach based on scalable video encoding and priority dropping. Without dynamically manipulating encoding parameters, a scalable encoding approach allows a wide range of video rates at the expense of some compression efficiency. Adaptation is supported in PPS by prioritizing data in the scalable video stream and dynamically dropping data in priority order in order to match the target bandwidth. PPS's sending strategy does not rely on complex control models and is independent of receiver feedback. Instead, it allows an underlying congestion control protocol, such as TCP or any of the TCP-friendly streaming protocols [9] to determine the appropriate sending rate. Whatever that rate is, PPS sends video packets in priority order from a window as fast as possible. In this way, a high-bandwidth receiver gets more data than a low-bandwidth receiver for each window. They both get the best possible video quality under their bandwidth limitations because for either receiver, the data packets received are more useful than those discarded, and the maximum possible bandwidth is used while preserving TCP-friendly behavior. A key advantage of this approach is the simplicity of the mechanisms and the ability to support heterogeneous simulcast distribution efficiently.

3. Priority-progress streaming

3.1. Basic streaming

PPS uses timestamps and priority labels to perform adaptive streaming. A window in PPS contains all data packets with timestamps within a certain period of time. The window is called an adaptation window; and the adaptation window size is the time duration, not the number of packets or number of bytes in this window.

Figure 1 shows an ideal example of PPS streaming with sufficient bandwidth and a constant delay. Data packets with timestamps and priority labels are grouped into windows in time order. Suppose the timestamps are in milliseconds, and the window size is 100ms. Within each 100-ms window, packets are sorted and sent in priority order, assuming that a small number represents a high priority. Packets in a window are sent out as fast as possible. Hence, when PPS runs over TCP, it can deal with TCP's burstiness. In this example, the bandwidth is higher than the data rate, so there is spare time in the 100-ms window. The spare time can be used for work-ahead or bandwidth skimming [5].

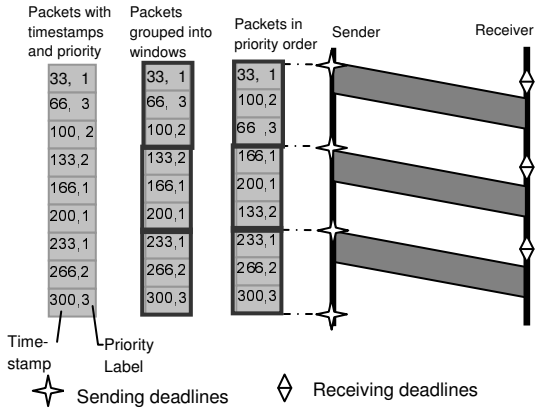

Figure 1. PPS streaming

3.2. Adaptation

As shown in Figure 2, PPS can adapt to the available bandwidth. If the bandwidth is lower than the data rate, some data packets are unsent when the window time expires. These data packets, which have low priorities, are dropped. PPS makes efficient use of the limited bandwidth by transferring the data packets with highest priority first.

Figure 3 (a) shows how PPS deals with increased delay by asking the sender to send data earlier so that it has more time to reach the receiver. If the delay decreases, PPS could either change back to the old sending schedule to keep the receiver buffer fill-level low, or keep the current schedule so as to prime the receiver-side buffer in anticipation of future delay and bandwidth variations.

In practice, the two adaptation mechanisms cooperate to match the varying network conditions.

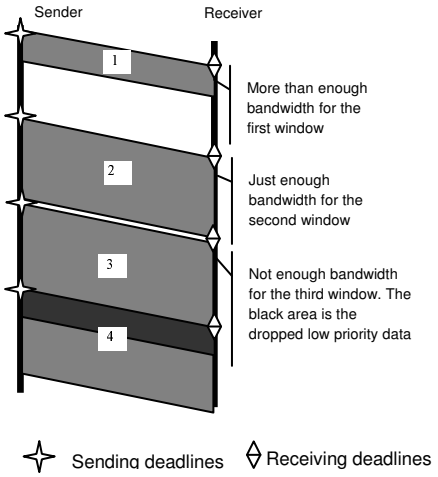

Figure 2. Adaptation to bandwidth

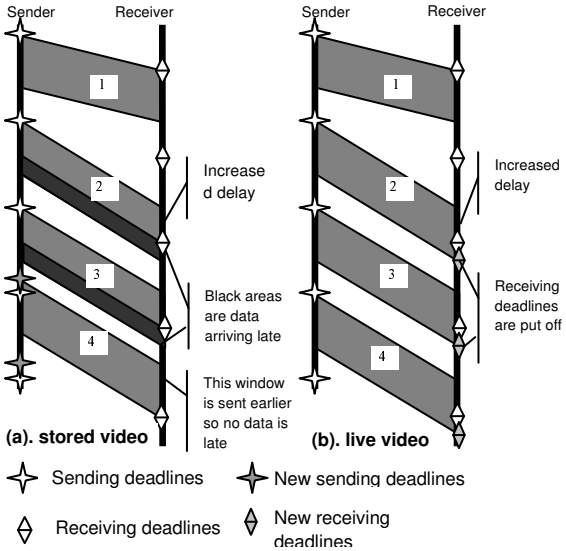

Figure 3. Adaptation to delay

3.3. Preparation for video streaming

PPS can be used to stream any data flow that can be packetized such that each packet can be time-stamped and prioritized. In this section, we discuss packetization, timestamping, and prioritization for video streams.

A video stream consists of video frames; the video frames could be the data packets for PPS. However, how the video frames are encoded determines the space for adaptation. Scalable encoding is preferred because the video stream can work at different data rates and we can achieve different quality levels under different network conditions. The Quasar pipeline uses a scalable compression format called SPEG (Scalable MPEG), extending MPEG-1 video with SNR scalability [5]. Each MPEG video frame is divided into four layers, in which the base layer contains the most significant bits of the DCT coefficients and the successive layers contain the less significant bits. Each layer of an MPEG frame is encapsulated in an SPEG packet.

Video frames have inherent timestamps: the play time. SPEG packets are given the timestamps of the corresponding MPEG frames.

Prioritization enables PPS to do wise adaptation without understanding the complex semantics of video encoding. In the Quasar pipeline, prioritization exposes temporal scalability and SNR scalability by reflecting dependencies among SPEG packets. For example, the base layer of an I frame has higher priority than the base layers of any P frames that depend on it, and a base layer has higher priority than the enhancement layers in the same MPEG frame. However, dependencies decide only a partial order among SPEG packets. For the importance of the base layer of a P frame over an enhancement layer of an I frame, Quasar's prioritization mechanism takes into account how much a user prefers frame rate over picture SNR. This mechanism does not simply assign a priority according to the frame type and layer; instead, it uses a window-based scheme resulting in many more priority levels for a video stream than a one-frame-based algorithm and hence supports much finer-grain adaptation. The larger the window, the more priority levels can be utilized. We combine some quality levels that are indistinguishable by human eyes and we define at most 16 priority levels at any given time in the Quasar pipeline. Details of the algorithm can be found in our earlier papers [5].

SPEG is just an example scalable video format. Subsequently new scalable video encoding approaches, such as MPEG-4 FGS, have better compression efficiency and finer-grain scalability than SPEG and hence offer an even more favorable platform for PPS.

4. Live streaming

4.1. Adaptation for live video

Using PPS for live video introduces much more than simply replacing the stored video file with a video camera. A big difference between stored video and live video is that live video has its own capture clock. Hence, live video cannot be generated faster or slower than its capture rate, while stored video can be read whenever it is needed. This difference implies that the work-ahead mechanism described in Section 3.2 for dealing with increased delay cannot be used for live video. For live video, since it is not possible to "read ahead", it cannot be sent ahead; we instead introduce delay at the receiver by pushing back the receiving deadlines, as shown in Figure 3 (b).

Note that this mechanism is suitable for multicast because each receiver can adjust its receiving deadlines to compensate for its own network delay.

4.2. Latency for streaming

The capture clock introduces the notion of end-to-end latency, which is the time from a frame being captured to its being displayed. Reducing this end-to-end latency is a goal specific to live video streaming. For stored video streaming, applications do require that frames arrive on time for display, but it does not matter when the frame is read from a file or how long it stays in buffers as long as it is on time for display.

There are two main sources of latency: the end machines and the network. Latency from the end machines includes the processing time and the buffering time. On both the sender and the receiver, the processing time does not vary much. For example, the time for encoding, decoding, reordering, and prioritizing is fixed unless we improve the algorithms or switch to faster computers. Therefore we can assume that in general these times are fixed.

Two types of buffers contribute to the total buffering time. Some buffers enable asynchrony among pipeline components. For example, the capture buffer keeps raw video frames from being dropped while the CPU is occupied by encoding or prioritizing; similarly the display buffer allows a smooth playback when a complex frame takes longer than its display duration to decode. These buffers need only be large enough to prevent the pipeline from stalling. The other type of buffer permits adaptation. The time spent in these buffers is determined by the adaptation window size, which is an adjustable PPS parameter.

The latency of the network segment is something that we adapt to and cannot control.

Ignoring the latency sources that are independent of PPS, the end-to-end latency due to adaptation buffers is the sum of the adaptation window size and the transmission time, as shown in Figure 4. For PPS adaptation, the last packet in the window is delayed on the sender side for the whole window time but not delayed at all on the receiver side. Similarly, the first packet in the window is delayed on the receiver side but not at all on the sender side. All of the frames in between are delayed both on the sender and the receiver, but the total delay is always one window time. The transmission time is related to the window size in two ways. When the bandwidth is higher than the data rate, the transmission time is proportional to the amount of data in a window, which is proportional to the window size. When the bandwidth is lower than the data rate, the window size is the transmission time for this window, according to the PPS streaming algorithm (since transmission continues for the entire duration of the window before data is dropped).

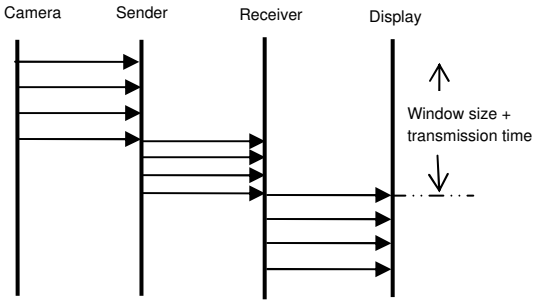

Figure 4. End-to-end latency

In summary, for the normal case when bandwidth is limited, the latency inherent in PPS is generally twice the window size. Thus, tuning the adaptation window size is the key to tuning the end-to-end latency. For low latency streaming, a small window size is preferable. However, a small window size makes fine-grain adaptation difficult and eventually impacts video quality. This is because adaptation happens within a window, i.e. a smaller window provides fewer droppable data units and fewer priority levels for adaptation.

5. Experiments

We have implemented the live Quasar pipeline by extending PPS for live streaming and substituting the MPEG source and the SPEG transcoding components of the pipeline with a camera, a capture card, and a software SPEG encoder. The capture card we use is a WinTV card from Hauppauge. The SPEG encoder is based on ffmpeg [2], an open source encoder that can encode in real time. We modified ffmpeg to implement SPEG's SNR layering strategy and to produce SPEG output directly to the live Quasar pipeline.

The live Quasar pipeline runs on Linux Mandrake 8.1. The sender and the receiver are two Pentium III 930MHz machines. The transport protocol we use is TCP. We run the pipeline on a private LAN without any competing traffic. We also maintain minimum buffer fill levels that allow the pipeline to run smoothly. Thus the adaptation window size is the main control variable in the experiments.

Measurements are obtained through the gscope software oscilloscope [3], which is a time-sensitive visualization tool that shows the bandwidth usage, buffer fill level, end-to-end latency, and other signals in real time.

In the following subsections, we concentrate on the relationships between end-to-end latency, the adaptation window size, and the adaptation granularity. We use the adaptation granularity as an indication of the effectiveness

of the adaptation. The adaptation granularity determines how closely a pipeline can utilize a given level of resource capacity, which is bandwidth in our experiments.

5.1. Latency vs. window size

Figure 5 shows the relationship between the latency and the adaptation window size. As expected, the latency grows with the window size. From Figure 5 we can see that when the adaptation window size is less than 167ms, the latency from adaptation, plus processing and necessary buffering, is well below 400ms. In the real world, the end-to-end latency also includes the network propagation delay and transmission time.

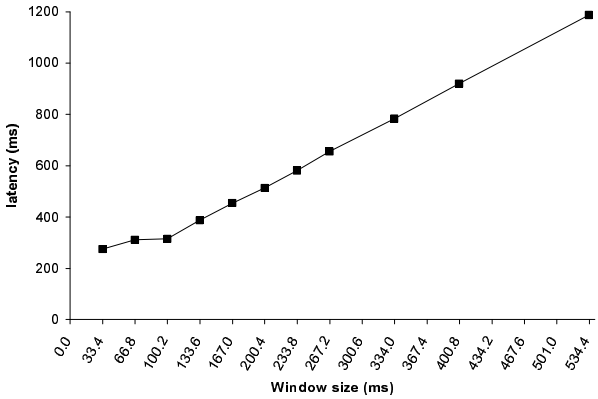

Figure 5. Latency vs. adaptation window size

The latencies shown in Figure 5 are measured for an intra-encoded video stream. The latencies for inter-encoded streams are very close to those shown in Figure 5 and the GOP size has little impact on the end-to-end latency. The window size is the determinant factor.

5.2. Adaptation granularity vs. window size

Each window size can deliver a certain number of possible quality levels. These quality levels range from full quality, when all packets of the window are delivered, to zero quality when none is delivered. Between these two extremes lie a number of quality levels, one for each priority, whose bandwidth requirements can be represented as a percentage of the full quality video bandwidth. As discussed in Section 2.3, the number of priority levels and their corresponding bandwidth percentages depend on the scalability of the video stream, the window size, and the user preferences.

In Figure 6, Figure 7, and Figure 8, we show samples of quality levels available for different window sizes and user preferences. Each symbol + in the plot area represents a quality level. The x value of the symbol is the window size in which that quality level is available; the y value of the symbol is the percentage of the full quality video bandwidth for that quality level. Figure 6 shows the available quality levels when a user prefers temporal quality and SNR quality equally; Figure 7 shows the available quality levels when a user prefers the maximum temporal quality; and Figure 8 shows the available quality levels when a user prefers the maximum SNR quality.

Ideally, for each window size there should be many available quality levels and their bandwidth percentages should be evenly distributed in order to closely match the varying network bandwidth. However, the scalability of video encoding and the window size determine how many prioritizable and independently droppable units are in a window and the sizes of these units determine the distribution of bandwidth percentage for quality levels. For the window size of 33.4ms, each window includes only one MPEG frame at NTSC rate. If no scalability is introduced, there is only one droppable unit in the window and there is only one quality level whatever the user preference is. If we double the frame rate (or double the window length), we introduce some temporal scalability and there are two MEPG frames in the window thus two droppable units and two quality levels. If we introduce SNR scalability into MPEG by using SPEG encoding there are four quality levels hence four droppable units per frame.

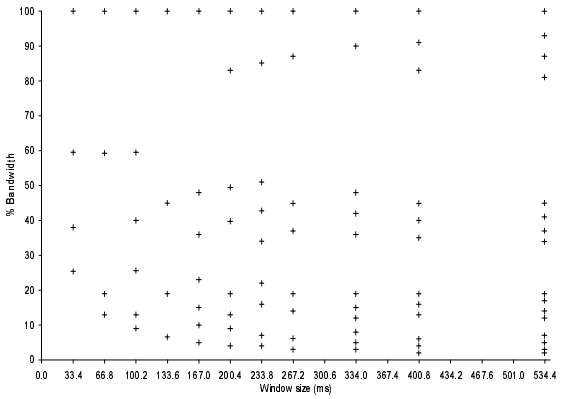

Figure 6. Adaptation granularity vs. window Size (neutral preference)

In order to minimize latency, we need to minimize the window size while maintaining a large enough number of evenly distributed quality levels to enable fine-grain adaptation. For SPEG, a window size of 133.6ms seems to be a good choice, since it has more than 10 quality levels and allows the pipeline to achieve relatively low, less than 400ms, total end to end latency. However, SPEG has only two dimensions of quality adaptation, the temporal adaptation and the SNR adaptation; and the SNR adaptation is relatively coarse-grained. Thus, any results obtained with SPEG could easily be improved with scalable video encodings that provide finer granularity

scalability. With improved scalable video encoding, PPS could easily support interactive streaming with a latency requirement of under 200ms.

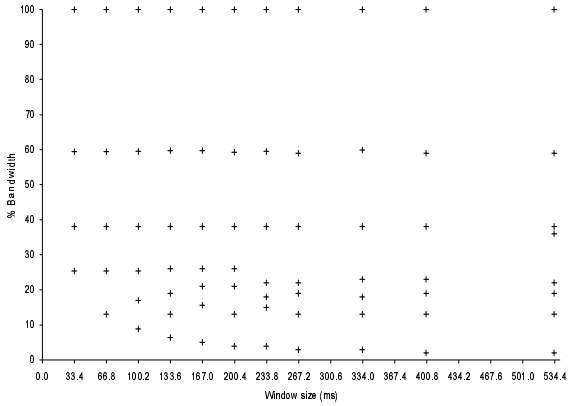

Figure 7. Adaptation granularity vs. window size (prefer temporal quality)

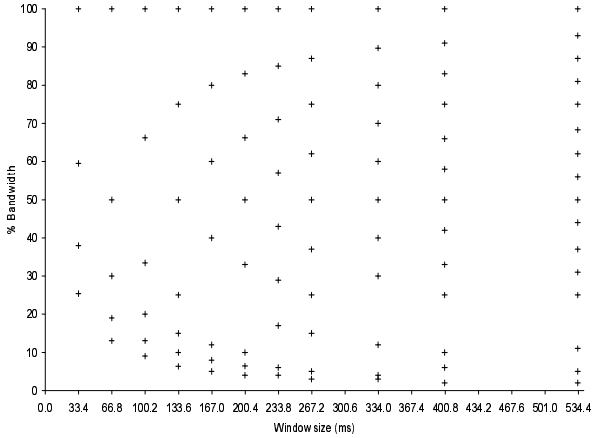

Figure 8. Adaptation granularity vs. window size (prefer SNR quality)

6. Conclusion and future work

Priority progress streaming is a generic and efficient mechanism for fine-grain adaptive streaming of stored media. However, it implies increased latency for reordering data into priority order prior to transmission and for reordering back into time order after transmission. In this paper we explored the real world impact of this reordering latency for live-source video pipelines. We showed that even using a coarse-grained scalable video encoding approaches reordering latency can be reduced to under 400ms, making the approach applicable to many video surveillance applications. As finer granularity video encodings become available, the same level of fine-grain adaptivity will be available using even smaller reordering windows, and interactivity will be easily supported using PPS.

In the future, we plan to extend PPS for multicast delivery and to build a many-to-many video surveillance infrastructure using it.

7. References

[1] J.-C. Bolot and T. Turletti. “Experience with control mechanisms for packet video in the Internet”. *ACM SIGCOMM Computer Communication Review*, vol. 28, pp. 4--15, January 1998.

[2] http://ffmpeg.sourceforge.net/

[3] A. Goel and J. Walpole. “Gscope: A Visualization Tool for Time-sensitive Software”. In *Proceedings of the Freenix Track of the 2002 UNSENIX Annual Technical Conference*, June 2002.

[4] S. Jacobs and A. Eleftheriadis. “Streaming Video Using Dynamic Rate Shaping and TCP Congestion Control”. *Journal of Visual Communication and Image Representation*, 9(3), 211-222, 1998.

[5] C. Krasic and J. Walpole. “QoS Scalability for Streamed Media Delivery”. *CSE Technical Report CSE-99-011,* Oregon Graduate Institute, September 1999.

[6] C. Krasic and J. Walpole. “Priority-Progress Streaming for Quality-Adaptive Multimedia”. In *Proceedings of the ACM Multimedia Doctoral Symposium*, Ottawa, Canada, October 2001.

[7] C. Krasic, J. Walpole, and W. Feng. “Quality-Adaptive Media Streaming by Priority Drop”. To appear in *NOSSDAV 2003*, June 2003.

[8] H. Liu and M.E. Zarki. “Adaptive Source Rate Control for Real-time Wireless Video Transmission”. *Mobile Networks and Applications*, 3:49--60, 1998.

[9] R. Rejaie, M. Handley, and D. Estrin. “RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet”. In *Proceedings of IEEE INFOCOM*, volume 3, page 1337-1345, New York, NY, March 1999.

[10] B. Vandalore, W. Feng, R.Jain, and S. Fahmy. “A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia”. *Journal of Real Time Systems (Special Issue on Adaptive Multimedia)*. January 2000.

[11] D. Wu, Y. T. Hou, W. Zhu, Y-Q Zhang, and J. M. Peha. “Streaming Video over the Internet: Approaches and Directions”. *IEEE Transaction on Circuits and Systems for Video Technology*, VOL. 11, NO. 3, March 2001.