Bamba—Audio and video streaming over the Internet

by M. H. Willebeek-LeMair
K. G. Kumar
E. C. Snible

The World Wide Web has become a primary means of disseminating information, which is being presented increasingly through multiple media. The ability to broadcast audio and video information is becoming a reality with the advent of new media-streaming technologies. Most of the emerging streaming systems require high-bandwidth connections in order to deliver audio and video of suitable quality. In this paper we present a media-streaming system, called Bamba, that delivers audio and video over low-bandwidth modem connections with the use of standard compression technologies. Bamba offers high-quality audio and video over low-bit-rate connections and can operate using a standard HTTP server. The Bamba video is enhanced with special provisions for reducing the effect of errors in a lossy-network environment. Bamba adheres to existing standards wherever possible. Finally, Bamba has been fully implemented and deployed both internally at IBM and externally.

1. Introduction

The World Wide Web (WWW) has become a primary means of disseminating information. Initially, the type of information distributed was primarily in the form of text and graphics. Later, images and stored audio and video files emerged. These audio and video files are downloaded from a server and stored at the client before they are played. Most recently, streamed audio and video have become available from both stored and live sources on the Web. Audio and video streaming enables clients to select and receive audio and video content from servers across the network and to begin hearing and seeing the content as soon as the first few bytes of the stream arrive at the client. Streaming technology involves audio and video compression, schemes for stream formatting and transmission packetization, networking protocols and routing, client designs for displaying and synchronizing different media streams, and server designs for content storage and delivery. In this paper we present a system for audio and video streaming (with code name Bamba) developed at the IBM Thomas J. Watson Research Center. Bamba has been deployed within IBM and was demonstrated externally on the official Web site of the 1996 Olympics. It has since been made available for free download from the IBM AlphaWorks* Web site.[1]

Today's computer-network infrastructures, including the Internet, were not designed with streaming in mind. Streaming media requires that data be transmitted from a server to a client at a sustained bit rate that is high enough to maintain continuous and smooth playback at the receiving client station. A primary objective in developing Bamba is to stream audio and video across the Web through very-low-bit-rate connections. Audio is

[1] *http://www.AlphaWorks.ibm.com*

0018-8646/98/$5.00 © 1998 IBM

sufficiently compressed to stream over modem connections at 14.4 Kb/s, and video at 28.8 Kb/s. The system that has been developed not only achieves the low-bit-rate goal, but can also be extended to support higher-bit-rate streams to provide higher-quality streaming over intranets or higher-bandwidth Internet connections. Furthermore, when streaming is not possible because of congestion or insufficient bandwidth availability, the Bamba player (client software) at the receiving client automatically calculates how much data to preload in order to maintain continuous playback. This allows clients connected via low-bit-rate connections to fall back to a download-and-play mode and still receive the higher-bit-rate content.

Existing audio and video streaming technologies

In recent years, there has been much research and development in the areas of audio and video streaming as well as videoconferencing. Videoconferencing differs from audio and video streaming in that the communication is bidirectional, and end-to-end delays must be very low (<200 ms) for interactive communication. In fact, videoconferencing standards are quite mature and have emerged from the International Telecommunication Union (ITU) in the form of the H.3xx standards [1, 2], and from the Internet Engineering Task Force (IETF) in conjunction with the multicast backbone (MBone) [1, 3, 4]. In general, the two camps use the same audio and video compression standards (defined by the ITU) but differ in their networking protocol specifications.

Audio and video streaming differs technically from its videoconferencing counterpart in that it can afford greater flexibility in end-to-end delays when the data is transmitted across a network and in the fact that stored content may be manipulated off-line with additional processing. These begin to merge when one considers live audio and video streaming applications (e.g., Internet, radio, and TV). The most relevant of the ITU standards is H.323, which defines audio/visual services over LANs for which quality of service cannot be guaranteed [5]. This standard specifies a variety of audio and video coders and decoders (CODECs) as well as signaling protocols to negotiate capabilities and set up and manage connections [6]. The underlying transport specified is the Real-time Transport Protocol (RTP) [7]. This protocol, defined by the IETF, is intended to provide a means of transporting real-time streams over Internet Protocol (IP) networks. A new protocol, the Real Time Streaming Protocol (RTSP), just proposed to the IETF, more directly addresses the issues of delivering and managing multimedia streams [8]. Clearly, this area is still evolving as new protocols are being defined and refined to satisfy a wide range of emerging networked multimedia applications.

There are a large number of audio and video streaming systems available in the market today [9]. These include VDOLive**,[2] StreamWorks**,[3] Vosaic**,[4] VivoActive**,[5] InterVU**,[6] and RealAudio**.[7] VDOLive, Streamworks, Vosaic, and RealAudio are based on proprietary client–server systems that transport their audio and video streams by means of User Datagram Protocol (UDP/IP) connections. This unreliable transport does not retransmit lost packets and is blocked by most firewalls unless they are specially reconfigured. The others use HTTP (based on TCP/IP) [10]. VDOLive employs a proprietary hierarchical compression technique that allows the server to adapt the video-stream bandwidth to the available network connection bandwidth. StreamWorks, Vosaic, and InterVu are based on MPEG** [11], while Vivo uses H.263 [12]. In general, these systems are designed to work over higher-bandwidth LAN connections and not at modem speeds. At modem speeds, the MPEG-based systems revert to slide-show-type video.

Bamba is a streaming system that was designed to run over existing computer network infrastructures. In particular, it is versatile in dealing with the heterogeneous nature of this environment and the unpredictable congestion behavior of today's network traffic. In the Bamba system, audio and video are compressed into a Bamba file. This file is specially formatted to interleave the audio and video content and may even be extended to include other data types. The Bamba file is placed on a server. A client equipped with the appropriate Bamba software is able to communicate with the server and receive the Bamba audio/video file. If the network conditions are suitable (sufficient sustained bandwidth is available), this file, streaming across the network, is played at the client immediately. Otherwise, the file is played once uninterrupted playback can be ensured.

The Bamba streaming system has several key features. The first of these is the quality of the audio and video, where the audio is set at a constant 6.3 Kb/s and the video ranges from very low bit rates of tens of kilobits per second to hundreds of kilobits per second. The second is the fact that both the audio and video compression are based on standard algorithms and can be performed by standards-compliant decoders. Third, the Bamba streaming system uses either a standard HTTP server or an enhanced video server running RTP over UDP/IP. In the HTTP case, no special server software is required to store and send Bamba clips, and the transmitted streams can traverse firewalls with no special firewall configuration requirements. In the case of the video server running RTP

[2] *http://www.vdo.net*
[3] *http://www.xingtech.com*
[4] *http://www.vosaic.com*
[5] *http://www.vivo.com*
[6] *http://www.intervu.com*
[7] *http://www.realaudio.com*

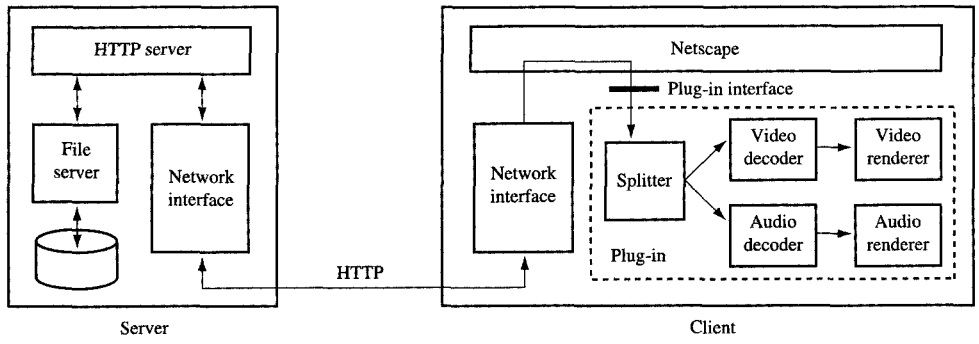


Figure 1

Bamba system block diagram.

over UDP/IP, additional functionality is provided by means of a control protocol between the client and server. This functionality includes pacing of the transmission stream at a target bit rate as well as specific start and end times of transmission within an audio or video file. Finally, the Bamba player has been implemented either as a helper application, which runs outside a Web browser, or as a browser plug-in, which enables application developers to embed audio and video clips easily within an HTML document or as a Java** applet, which can be downloaded directly from a Web server containing Bamba clips without requiring special software installation at the client.

The rest of this paper is organized as follows. In Section 2, we describe the underlying Bamba technology. This includes a description of the video-compression algorithm as well as details related to the overall system design. In Section 3, we describe several enhancements made to the basic Bamba streaming system, such as increased robustness in lossy-network environments. A description of the Live Bamba architecture is given in Section 4. The paper is summarized in Section 5.

2. Bamba technology

A base requirement of the Bamba streaming system is to function within the WWW standard HTTP-based client–server architecture. In this section, we provide a description of the overall client–server architecture and present details concerning the compression algorithms. We also describe the Bamba file format and synchronization technique.

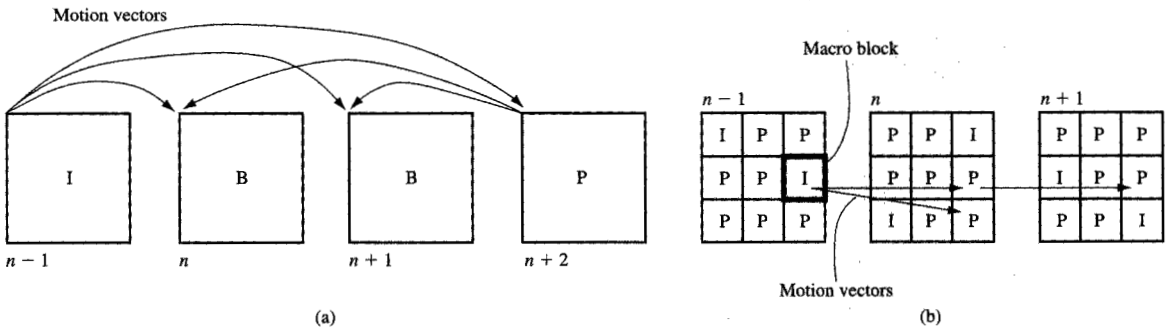
- *Bamba streaming architecture*

A block diagram of the Bamba streaming system is presented in **Figure 1**. The system consists of a client and a server component. The server is a standard HTTP Web server, which contains the stored Bamba audio and video files. The client consists of a Web browser and the Bamba audio and video plug-in software.

The Bamba plug-ins are implemented as a set of dynamic link libraries that interface with the Web browser through the Netscape-defined plug-in API. Netscape has defined a set of plug-in routines that are used to communicate between the plug-in and the browser. Each plug-in library contains an initialization routine within which is declared what Netscape plug-in routines are used by the plug-in. These routines include mechanisms to create and delete instances of a plug-in, manage the plug-in display window, control the flow of data streams to the plug-in, etc. In general, the plug-in is tightly integrated with the browser. Note that while Netscape was used in this example, the approach is similar for other browsers.

Bamba files may be embedded in HTML pages by means of a URL pointing to a file on an HTTP or video server. When the URL is requested, the server passes the metadata identifying the Bamba file and containing information about the file type to the client. The file type is used by the browser to launch the appropriate plug-in to play back the Bamba file.

Bamba was designed to stream clips from standard HTTP Web servers without special streaming software on the server. As such, Bamba is limited to the communication mechanisms provided by the HTTP protocol. This approach has certain advantages, the greatest of which is that it is simple and maps gracefully into the existing Web browsing architecture. As a result, content creators can easily produce Bamba audio and video clips and embed them in standard HTML [13] pages, which are then loaded onto and accessed from a standard HTTP server. Since the underlying transport protocol used by HTTP is TCP/IP, which provides reliable



Video-compression algorithms: (a) MPEG I-, P-, and B-frame compression dependencies; (b) H.263 I and P macro-block dependencies.

end-to-end network connections, no special provisions are required for handling packet loss within the network. In essence, a Bamba audio or video clip is treated like any other HTTP object, such as an HTML or JPEG [14] file. If selected, the Bamba clip is transferred to the client (browser station) as fast as TCP/IP can move it, and the client begins decoding and displaying the Bamba file as soon as the first few bytes arrive.

Since Bamba uses TCP/IP as the underlying communication protocol, the streams can traverse firewalls with no special configuration requirements. In general, systems based on UDP/IP cannot traverse firewalls without explicit permission changes in the firewall to allow passage to the UDP/IP packets. This is because UDP/IP packets are easier to imitate than TCP/IP packets, since the UDP/IP protocol involves no end-to-end handshakes or sequence numbers [15].

• *Bamba audio and video technology*

The audio and video technology used in Bamba is based on standard algorithms originally defined within the ITU H.324 standard for video telephony over regular phone lines [16]. The audio standard, G.723, specifies two bit rates: 5.3 Kb/s and 6.3 Kb/s [17]. Bamba uses the higher-bit-rate CODEC, which compresses an 8-kHz input of 16-bit samples to a fixed 6.3Kb/s stream. This audio algorithm is optimized to represent speech at high quality over low-bit-rate connections. It encodes speech into 30-ms frames by means of linear predictive analysis-by-synthesis coding [17]. The input signal for the higher-bit-rate coder is Multipulse Maximum Likelihood Quantization (MP-MLQ) [17].

The Bamba video CODEC complies with the H.263 video compression standard [12], which uses an approach based on the discrete cosine transform (DCT). This is similar to the technology used for MPEG. Unlike MPEG, which uses intrapicture frames (I-frames), predicted frames (P-frames) and bidirectional predicted frames (B-frames), H.263 does not define I- and P-frames, but rather I- and P-blocks—8-pixel by 8-pixel subregions of a frame. **Figure 2(a)** illustrates the MPEG dependencies among I-, P-, and B-frames, while **Figure 2(b)** illustrates the partitioning of H.263 frames into I- and P-blocks and the dependencies between blocks. Representing frames as collections of I- and P-blocks reduces the size variance between frames and adds flexibility in selecting the refresh distance between I-blocks for different regions of the video image. To maximize compression based on temporal redundancy, there may be long intervals between I-blocks for regions in the image that are not changing.

The H.263 algorithm is designed to deliver video over very low-bit-rate (<64 Kb/s) dedicated connections. In this low-bit-rate range, H.263 has been demonstrated to outperform its predecessor, H.261 [18], by a 2.5:1 ratio [2] (i.e., at the same bandwidth, the signal-to-noise ratio of H.263 is 2.5 times higher than that of H.261. H.263 can also be easily extended to higher bit rates, in the 100–200-Kb/s range. These rates are suitable for streaming over ISDN or intranet LAN-type connections. The H.263 video compression algorithm uses a planar YVU12 format, which contains three components: luminance (Y) and two chrominance planes (V and U). The sizes of these planes vary as a function of the video resolution. Two of the resolutions supported by Bamba are the Common Intermediate Format (CIF) and the Quarter Common Intermediate Format (QCIF) [2]; the formats are presented in **Table 1**. Smaller and intermediate-size resolutions are also supported. The resolution and target bit rate are selected at compression time. The compression target bit rate may be set anywhere between 10 and 356 Kb/s.

As with many compression standards, the H.263 standard specifies the format of the video so that any standards-compliant decoder can successfully decode the video stream. Typically, this leaves much flexibility in the actual encoding technique and implementation. The H.263 encoding used for Bamba uses an innovative algorithm to trade frame rate for frame quality [19]. The art in video compression lies in the decision of how best to apportion a few bits to different components in the compression process so that the compressed stream, once decoded and displayed, produces the highest quality as perceived by the end user. Quality is highly ambiguous and is perceived differently by different users. A typical tradeoff is between frame rate and frame quality (pixel quantization). For the same number of bits, it is possible to create two very different standards-compliant streams. One stream may have a higher frame rate, while the other may have a finer quantization of the frame pixels, obtaining a sharper image.

The Bamba video implementation incorporates a dynamic frame-rate-control algorithm, which trades frame rate for frame quality (bits per frame) while maintaining a constant average bit rate. This approach allows the video to balance between the two extremes and deliver smoother motion or sharper images as appropriate, depending on the content and scene changes in the video. The algorithm behavior is illustrated in **Figure 3**. A video sequence with dynamically changing content is used to illustrate the algorithm's adaptable frame rate. The original clip is approximately 30 seconds long, captured at 15 frames per second for a total of 445 frames. It was compressed at a target bit rate of 20 Kb/s and resulted in a total of 332 frames. Typically, larger frames are followed by a drop in frame rate in order to maintain the constant bit rate. The spikes in the figure correspond to larger frames, generated when the scene changes or the amount of motion in a scene is significant. These spikes are typically followed by several frame periods in which no data is transmitted at all.

The Bamba H.263 implementation includes special motion-estimation techniques [20] and fast DCT algorithms [21, 22], which result in very efficient implementations.

• *Framing structure*

A simple framing technique for smooth playback was implemented. Audio and video are interleaved into a single file to simplify the server function. Essentially, the server treats a Bamba file as any other data file. Audio and video data are interleaved proportionately to maintain a synchronous playback of both streams at the client. Bamba frames consist of a 240-byte segment of audio and a $240\beta/\alpha$-byte segment of video, where α is the audio rate and β is the video rate.

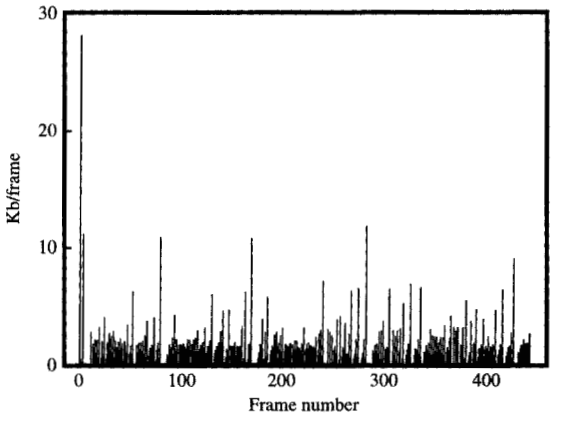


Figure 3

Illustration of Bamba video compression with dynamic-frame-rate-control algorithm. The number of bits transmitted is shown for each frame of a sample video sequence.

Table 1 CIF and QCIF planar YVU12 formats.

	Pixels/line	*Lines/frame*
CIF luminance (Y)	352	288
CIF chrominance plane (V)	176	144
CIF chrominance plane (U)	176	144
QCIF luminance (Y)	176	144
QCIF chrominance plane (V)	88	72
QCIF chrominance plane (U)	88	72

• *Streaming-control algorithm*

When the Web is accessed, the actual connection speed between a client and a server in the network varies depending on the access method (e.g., modem or LAN), the network load, the server load, and even the client load. Hence, it is rarely possible to guarantee performance in this "best-effort" environment, where processing and bandwidth resources are typically evenly distributed among all competing applications. Consequently, when an audio and/or video clip is accessed over the network, there is no guarantee that the resources (bandwidth and processing) are available to play the clip smoothly. To handle this situation, Bamba has a built-in rate monitor that dynamically evaluates the effective data-transfer rate (σ) of a selected audio or video clip and compares this to the specified bit rate ($\alpha + \beta$) for the clip, which is contained in the clip header. If the specified rate is less than the measured rate, the clip can be played immediately. If, on the other hand, the specified rate exceeds the measured rate ($\alpha + \beta > \sigma$), a fraction of the clip is buffered

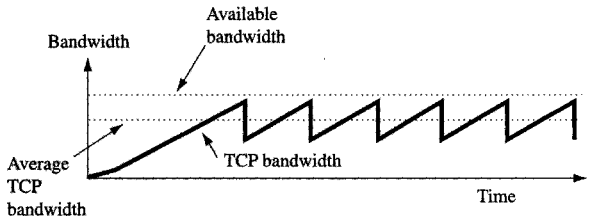


Figure 4

Illustration of TCP/IP bandwidth variation with time.

sufficient for the clip to play to completion smoothly once playback is started. The amount of prebuffering is $\delta = L[1 - \sigma/(\alpha + \beta)]$, where L is the clip length. This calculation is performed on the basis of the initial download rate and again any time the buffer underflows.

In future networks, where quality-of-service mechanisms will be able to guarantee a desired bandwidth, this approach will allow the clips to stream uninterrupted. It will also provide a simple means of characterizing the clips and making the appropriate bandwidth requests.

- *Synchronization technique*

To maintain synchronization between the audio and the video, a video interframe time is calculated as a function of the total number of video frames and the total length (in time) of the audio portion of the uncompressed clip. During compression, not all frames may be compressed, since some may be skipped in order to achieve the target bit rate. As a result, the compressed frames may not have contiguous frame numbers, so the spacing between frames is calculated as the difference in sequence numbers times the interframe time calculated earlier. Video frames are displayed on the basis of the video-frame sequence number, the interframe time, and the actual number of audio samples played. This approach is particularly powerful, since the actual video interframe time tends to vary depending on the capture hardware subsystem used to create the clip. Synchronization points may also be placed in the Bamba file in order to achieve playback at arbitrary points within a clip or to recover from errors during transmission when UDP/IP is used.

3. Bamba enhancements and error handling in a lossy environment

The HTTP client–server system has some limitations. First, the HTTP protocol has no explicit mechanisms to perform such sophisticated stream-control functions as seeking to a particular position in the stream. There are ways of carrying customized function calls within the HTTP stream, but this requires special server software to execute those functions. Second, TCP/IP is an inefficient protocol for streaming delay-sensitive data across the network. It was originally designed to transport data files, with a built-in mechanism to alleviate congestion in the network [23]. TCP/IP is based on a "sliding-window" protocol that waits for acknowledgments from the receiver for every packet it sends. Each packet in the sliding window has a timer associated with it. A packet-receipt acknowledgment must be received by the transmitter before the timer expires, or the packet will be retransmitted. The size of the sliding window (number of outstanding packets) is based on the speed with which acknowledgments are received. TCP/IP continues to increase the size of the window (effectively, the bandwidth at which it is sending) until packets start to time out. Once they time out, TCP/IP exponentially backs off (reducing the size of the sliding window) and retransmits these (presumably lost) packets. A typical TCP/IP bandwidth "profile" resembles a sawtooth, as shown in **Figure 4**, resulting in inefficient usage of the bandwidth.

The UDP-based solution is advantageous if sufficient bandwidth is available, since it does not use acknowledgments and allows the server to explicitly control the rate at which the streams are transmitted into the network. The continuous transmission at the server and the elimination of retransmissions make the resource requirements on the server much more predictable and manageable. On the other hand, this approach adds complexity to the server, since it must now pace the transmission of a clip into the network, and it adds complexity to the client side, since the client must be made able to handle packet loss within the network. However, for long clips, this approach reduces the storage requirements at the client and can provide a higher degree of functionality, such as the ability to seek and transmit only specified segments of the clip, or to adapt the transmitted bit rate to the available bit rate (e.g., send audio with no video or send selective portions of the video). Another important merit of UDP/IP is that, given the appropriate routing capability in the network, it can be used to efficiently multicast a stream to multiple clients simultaneously.

In the UDP-based Bamba system, the server store clips and has data pumps that pace the transmission of the video clips into the network. The system block diagram is similar to that of Figure 1, except that the network interface module, which receives the RTP/UDP packets from the network, makes sure they are presented to the splitter module in the correct order and, upon detection of a lost packet, resynchronizes the stream by searching for a new synchronization point.

• *Compression technology*

The H.263 video compression scheme was enhanced for Bamba, in order to provide added robustness to reduce the effect of lost packets in the UDP/IP environment [24]. Since a large percentage of each video frame within an H.263 compressed stream is encoded by means of P-blocks with interframe dependencies, corrupted data may create errors that propagate for extended periods until an I-block refreshes the region. In general, this makes the video more susceptible to errors. To reduce the error effects, a novel scheme was devised for selecting when and where to place I-blocks within the compressed stream. The scheme is based on a two-phase compression strategy. The first phase of the compression strategy is needed to construct a dependence graph based on motion vectors between pixels in successive frames. [A motion vector is a pointer from a P-block in the current frame to an I- or P-block in the previous frame. These motion vectors are then used to determine the dependence count (the number of future blocks that may depend on a given block) of each block in a sequence of compressed frames.] The second phase selects which blocks to compress as I-blocks on the basis of the dependence count of each block in a sequence of compressed frames. This demonstrably improves the ability of the compressed stream to recover from errors and greatly reduces the time required to reconstruct the video image when an error occurs. The approach is standards-compliant, maintains a smooth bandwidth profile of the compressed stream (small variance in size between compressed frames), and causes only a slight increase in the overall bandwidth requirements.

A conventional H.263 encoder first partitions a video image into a set of blocks of 16 pixels by 16 pixels. The coding control function searches for the best match between each block in the current frame and blocks in the previous frame. If a sufficiently close match is found, the block is encoded as a P-block based on the difference between the block and the closest matching block in the previous frame. The closeness of the match is evaluated in terms of the number of bits needed to encode the difference between the block and the closest matching block in the previous frame. If the difference is too great, it is deemed more efficient to encode the block independently, without reference to previous data. Such a block is referred to as an intracoded I-block.

The resulting H.263 compressed stream consists primarily of P-blocks, with I-block insertions caused by scene changes or severe motion. To prevent error accumulation, the standard also requires that each block be encoded as an I-block at least once every 132 frames. Although the H.263 standard defines how I-blocks and P-blocks are encoded, it allows considerable flexibility in selecting when to encode a block as either an I- or P-block. We exploit this flexibility to improve the robustness of a standards-compliant stream by carefully choosing when and where to insert I-blocks during the encoding process.

I-block encoding exploits only the spatial redundancy within the block in the compression process, while P-block encoding exploits both the temporal and spatial redundancies of the video. Although interblock encoding generally achieves more compression gain, the encoding dependencies of P-blocks reduce their resilience to errors. If the region referenced by a P-block has been corrupted, the decoding of the P-block will generate incorrect pixel values. If all or a part of this corrupted P-block is again referenced by other P-blocks, the erroneous pixel values will cause errors to propagate from one frame to another. This is known as the error-propagation problem of motion-compensated video compression. The propagation stops when all corrupted regions are updated by I-blocks.

On the Internet, where loss is primarily attributed to network congestion, the loss of a UDP/IP packet that contains video data can result in the loss of several frames in a low-bit-rate video system. For example, with a target bit rate of 20 Kb/s, a typical QCIF compressed video frame may contain 165 bytes. Hence, a 500-byte IP packet contains roughly three frames of compressed video data. If the packet is lost, these frames cannot be recovered, and errors begin to propagate.

For non-real-time applications, knowledge about the interdependence among blocks in a sequence can be obtained from the dependencies reflected by the motion vectors. It is thus possible to assign a measure of importance to a pixel or block by counting the number of pixels or blocks that depend on it. This operation is anticausal, i.e., traversing backward in time. The higher the dependence on a block, the more critical it is that this block be correct and that it be encoded as an I-block. Furthermore, dependence chains may be broken by encoding intermediate blocks in the chain as I-blocks.

We illustrate how to construct a dependence graph and calculate dependence counts with the example of three frames in **Figure 5**. By starting with the last frame in a sequence, N, and tracing the motion vectors between frames, one can determine the dependence on pixels in the previous frame. In Figure 5, pixels F and G in frame N refer to pixel D in frame $N - 1$, since the motion vectors from F and G point to D. Similarly, pixels H and I refer to pixel C. We define the dependence count of pixel D to be 2; i.e., two pixels in subsequent frames refer to D. Pixel A in frame $N - 2$ has a dependence count of 6 because pixels C, D, F, G, H, and I depend on it. Similarly, B has a dependence count of 3. If A is corrupted, the error spreads from one pixel to four pixels in just two frames. The graph formed by the pixels and motion vectors is a directed tree, which we call a dependence tree.

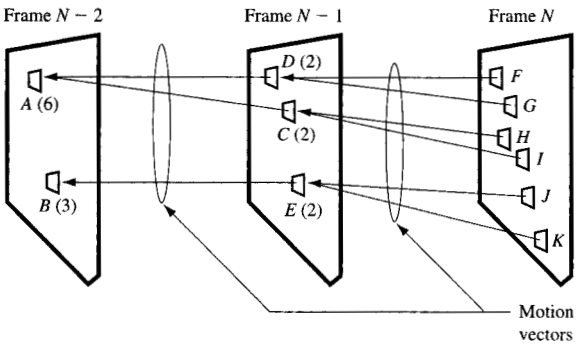


Figure 5

Video-compression motion-vector dependence graph.

Our I-block selection procedure sets a target I-block count, M, for the number of I-blocks to be inserted in a sequence of N frames. This target value depends on the desired level of robustness to packet loss and/or speed with which a client can produce an error-free decoding of a video stream when connecting to an ongoing session in midstream (referred to as join latency). The number of I-blocks also affects the frame rate in a fixed-bit-rate environment. In general, the more I-blocks inserted, the more robust the video, but the lower the number of frames that can be generated. To begin, an arbitrary dependence threshold is selected, representing the maximum number of pixels that may depend on the pixels in a block. Should a block's dependence count exceed the threshold, it is converted to an I-block. When a block is converted to an I-block, the dependence graph is segmented. After a complete iteration through the video sequence, the total number of I-blocks may be above or below the target. The algorithm then adjusts the threshold accordingly and reiterates until the target, M, is reached or the maximum number of iterations allowed is exceeded.

The H.263 standard specifies a forced update period (FUP) of 132 frames within which each block must be updated (i.e., coded as an I-block) at least once. In our scheme, the FUP can be made a function of the network packet-loss characteristics and the speed with which the video must recover from a corrupted state. In the case of packet losses, it is desirable that the decoded stream recover fully from a packet loss before the next loss occurs. Assume that each packet contains K frames of compressed video. For a packet loss frequency of one in every P packets, the recovery has to be completed within $K(P - 1)$ frames. On the basis of our experimental results, it takes approximately eight times the FUP to fully restore a video to an error-free state. One can thus calculate the maximum FUP to be approximately $K(P - 1)/8$ frames. For a 1% packet loss rate ($P = 100$) and $K = 3$, the maximum FUP is 37 frames. The other factor affecting the choice of the FUP is the join latency of video multicast. Let us assume a join latency of ten seconds. This is similar to the situation in which the decoded sequence must recover from a packet loss in ten seconds. If the video is streamed at approximately 15 frames per second, the maximum FUP is 18 frames [(10 * 15)/8]. The target I-block count, H, can be calculated as $H = (N * S)/FUP$, where S is the number of blocks in a frame. In this example, the FUP would be set at 18 frames in order to meet both the packet-loss recovery interval and join-latency requirements. The sensitivity of this parameter requires further study, but in practice we have chosen a set value of 20 that appears to produce reasonable results.

In certain cases (e.g., limited amount of memory), it might not be feasible to process an entire video before starting to encode it. With a little modification, the same algorithm can operate on a segment-by-segment basis. A video can be partitioned into non-overlapping segments, each segment being treated independently. The encoder performs the two-phase compression on all frames in a segment and then moves to the next, non-overlapping segment and encodes it independently. This technique cannot be applied to delay-critical applications when frames cannot be prebuffered. A scheme to deal with real-time applications is discussed in [20].

• *Bamba stream format and synchronization*

The Bamba stream format was designed with the UDP/IP environment in mind, where it is assumed that packets can be lost at any time during the transmission and that new clients may join a multicast transmission at any point during an ongoing broadcast. This makes it necessary that the packetized stream contain sufficient information for a client to interpret a packet's content at any point in the stream. This can be done by either a) packetizing the audio and video on distinct audio and video frame boundaries, so that the splitter module can automatically separate audio from video, or b) providing a means to detect the frame boundaries within the stream. The first approach adds overhead to the server, since the server must be aware of the audio and video frame boundaries and perform special packetization. The Bamba file format is equivalent to the streaming format; i.e., the server does not have to process the data in the file, but must simply place contiguous chunks of data from the file into packets and send them. For this reason, Bamba is implemented using the second approach, in which every Bamba file is separated into Bamba frames, each containing a header, a portion of audio, and a portion of video. Each Bamba frame header contains a unique synchronization symbol

(bit pattern) that does not reappear within the data (audio and video) portions of the stream. This way, whenever a packet is lost, the client begins searching the incoming bit stream until it detects the unique synchronization symbol. To ensure that the header synchronization symbol does not occur within the stream, the audio and video data are run through a byte-stuffing filter that alters the byte sequence in the event that the special symbol is detected in the input stream. The stream must be destuffed at the receiver. Sufficient information is provided within each Bamba frame header for the client to interpret how the audio and video data are apportioned within the variable-length Bamba frame and to initiate synchronization of the two.

The unique synchronization symbol consists of 4 bytes: 00 00 F0 F0. The byte-stuffing algorithm scans the stream in search of the pattern 00 00 F0. If the pattern is found, the algorithm inserts an FF. The destuffing routine searches for 00 00 F0. If it finds an FF following the sequence, it removes it; if it finds an F0, it recognizes the header-synchronization symbol (any other byte would represent an error). This guarantees that the synchronization symbol, 00 00 F0 F0, can occur only within the header.

The synchronization between audio and video is severely complicated when the possibility of lost packets exists. Basically, video is synchronized to audio. This is done by including audio and video segments within a Bamba frame so that the start of each audio segment corresponds to the start of the corresponding video segment. Hence, each Bamba frame contains an integral number of video frames along with the corresponding audio segment.

Bamba streaming technology has been used extensively on a variety of Web sites both within and outside IBM. For example, it was used on the official 1996 Olympics Web site to distribute audio and video clips concerning the games. For that event, approximately 100000 users installed the Bamba plug-in and played clips from the "Sights and Sounds" pages of the Web site. Clips were offered at two target bit rates: 24 Kb/s for modem-connected users and 100 Kb/s for ISDN- and LAN-connected users. More recently, a variety of education and training applications have been developed that incorporate Bamba. These include Web-based courses that combine graphics, course maps, discussion forums, and Bamba audio and video files, as well as seminar applications that deliver choreographed presentations of audio streams with graphics and text in a Web browser.

A video jukebox service was created for IBM internal use to distribute audio and video content over the Web. Content providers can send their content to a multimedia laboratory, where it is converted to Bamba files at several target bit rates. The Bamba files are placed on a Web server, and the URL pointers to the files are sent back to the content providers for their use. The service has seen a steady increase in usage; it has been used by corporate communications personnel to distribute important messages to IBM employees. For example, recently one of these clips, of 7.5 minutes duration, was placed on the server. Over a three-day period there were 21000 viewings of the clip, with a 6:1 preference for the higher-bit-rate version. At the peak, there were 10000 hits in one day and 1000 in one hour, with a peak bandwidth consumption of 7.6 Mb/s and an average bandwidth consumption of 2.2 Mb/s. We continue to monitor the streaming performance and characterize the audience viewing trends to best understand how the content is used and its impact on the networks over which it is delivered.

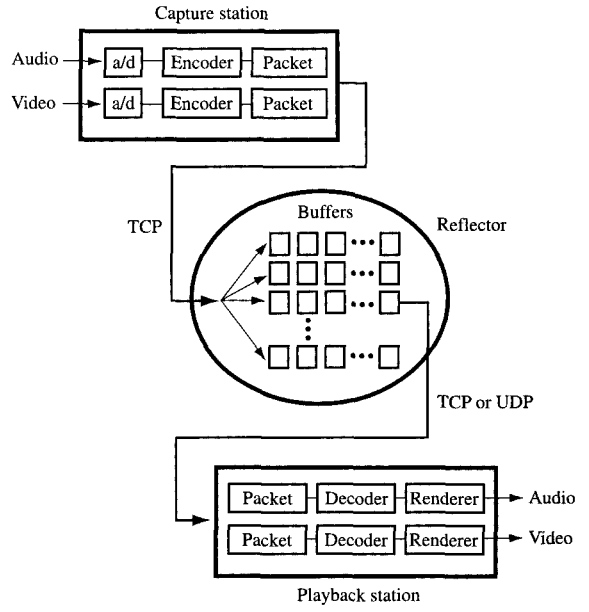


Figure 6

Live Bamba system block diagram.

4. Live Bamba architecture

A Live Bamba system was developed to stream audio and video from a live source across the Web to multiple recipients. This system uses the same audio and video compression technologies. The Live Bamba system consists of three primary components (as illustrated in **Figure 6**): an audio/video capture station, an audio/video reflector, and an audio/video playback station. In the capture station, audio and video inputs are converted from analog to digital form, compressed, and then packetized. The Live Bamba packets are transmitted to the reflector via a

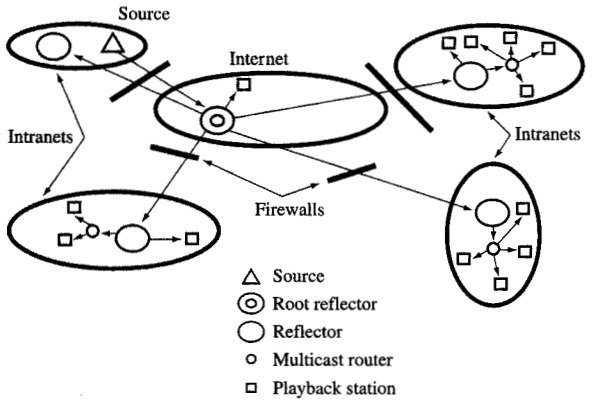


Figure 7

Hierarchical Bamba reflector configuration.

TCP/IP connection that is established between the reflector and the capture station. The reflector then establishes and manages multiple connections to interested recipients. These connections are initiated by the playback stations, which can either establish a direct connection to the reflector given the correct IP address, or establish a connection indirectly via a Web server through an HTTP URL, which returns a file to the playback station containing the appropriate address information and file type. The browser then launches the Live Bamba helper application or browser plug-in.

The reflector distributes the audio and video streams to the various playback stations in the network. Playback stations may join an ongoing session (live broadcast in progress) at any point in the transmission. The reflector maintains a circular buffer queue containing the most recent several seconds of a live transmission for each playback station to which it is connected. When a new station connects, the reflector produces a new copy of the circular buffer queue for that connection. Each of the circular buffers is written to by the incoming capture station input and read from by the TCP/IP connection to the corresponding playback station. The TCP/IP approach allows the connections to traverse firewalls easily and maintain high quality.

The same physical reflector node can be used for multiple sessions. The reflector node resource is limited, but upper bounds can be set for the number of connections per session as well as for the total number of connections per reflector. Furthermore, reflectors may be cascaded to scale and handle increased demand. A reflector may also be configured to provide multicast services when it is connected to networks with multicast capability. For example, point-to-point TCP/IP connections may be established between reflectors through firewall boundaries that separate intranets from the global Internet. Within the intranets, the reflector may establish multicast UDP/IP connections to local playback stations.

The reflector concept also provides a platform upon which customized features are easily built. For example, format conversions from high to low compressed bit rates to satisfy different network and playback-station capabilities are possible. It may also be preferable to maintain different audio tracks (e.g., different languages) for the same video feed and to route these audio tracks to different reflectors, depending on the reflectors' local audience preferences. A hierarchical Bamba reflector configuration is illustrated in **Figure 7**. In this example, a capture station in one intranet is transmitting to a "root" reflector in the Internet, which in turn is forwarding the signal to reflectors within different intranets. Within each intranet, the signal is multicast to local playback stations. Modifications to the streams could be made locally at each reflector.

5. Summary

The Bamba system for low-bit-rate audio and video streaming over the World Wide Web has been described, and a detailed description of the Bamba system design and significant issues related to its implementation have been discussed. Several key features distinguish Bamba from existing streaming technologies. The first of these is the quality of the audio and video, in particular the video, which ranges from very low bit rates of tens of kilobits per second to hundreds of kilobits per second. The second is the fact that both the audio and the video are based on standard algorithms. Third, the Bamba system can operate using standard HTTP servers. No special server software is required to store and send Bamba clips. Fourth, since Bamba uses TCP/IP as the underlying communication protocol, the streams can traverse firewalls with no special configuration requirements. Finally, the Bamba player was implemented as a Netscape plug-in, which enables application developers to easily embed audio and video clips within an HTML document. A UDP/IP-based system for Bamba has also been described, which requires a special server and supports multicast. For this system, a novel approach to enhancing the video robustness in lossy network environments has been presented.

The Live Bamba architecture was presented, which uses a reflector as the key building block to deliver the Bamba stream to multiple recipients. This approach is scalable and lends itself well to handling conversions of the stream at local access points, depending on local network and playback-station requirements and/or capabilities. Reflectors can provide secure communications and make use of network multicast capabilities when available.

Future systems will incorporate additional media types (e.g., graphics and Java applets) within the Bamba stream to produce synchronized streamed multimedia applications. Additional audio and video CODECs can easily be incorporated into the Bamba framework. As network technologies evolve, Bamba will make use of more advanced network-resource-reservation mechanisms, which will provide more control over the connections' quality of service.

Acknowledgments

We acknowledge the contributions of Yuan-Chi Chang, Zon-Yin Shae, Xiping Wang, and Steve Wood for their work on Bamba, and Marcel Kinard for the Bamba statistics regarding the video jukebox service. We also acknowledge the Science and Technology Group of IBM Research in Haifa for their work on the audio CODEC implementation, and the Multimedia Applications Group of IBM Research in Yorktown Heights for their work on the video CODEC implementation.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of VDONet Corporation Ltd., Xing Technology Corporation, Vosaic Corporation, Vivo Software Inc., InterVU, Inc., Progressive Networks, Inc., Moving Picture Expert Group, or Sun Microsystems, Inc.

References

1. M. Willebeek-LeMair and Z.-Y. Shae, "Videoconferencing Over Packet-Based Networks," *Research Report RC-20480*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1966.
2. Richard Schaphorst, *Videoconferencing and Videotelephony*, Artech House, Norwood, MA, 1996.
3. T. Turletti and C. Huitema, "Videoconferencing on the Internet," *IEEE Trans. Networking* **4,** No. 3, 340–351 (June 1996).
4. S. McCanne and V. Jacobson, "vic: A Flexible Framework for Packet Video," *Proceedings of Multimedia '95*, San Francisco, 1995, pp. 511–522.
5. "Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-Guaranteed Quality of Service," *Draft ITU-T Recommendation H.323*, International Telecommunication Union, Place des Nations, CH12-11 Geneva 20, Switzerland, May 1996.
6. "Control of Communications Between Visual Telephone Systems and Terminal Equipment," *ITU-T Recommendation H.245*, International Telecommunication Union, Place des Nations, CH12-11 Geneva 20, Switzerland, 1995.
7. "RTP: A Transport Protocol for Real-Time Application," Internet Draft, *draft-ietf-avt-rtp-05*, IETF Secretariat, c/o Corporation for National Research Initiatives, 1895 Preston White Drive, Suite 100, Reston, VA 20191, July 1994.
8. H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," *Internet Engineering Task Force MMUSIC WG Internet Draft*, IETF Secretariat, c/o Corporation for National Research Initiatives, 1895 Preston White Drive, Suite 100, Reston, VA 20191, February 2, 1998.
9. G. Venditto, "Instant Video," *Internet World*, pp. 84–101 (November 1996).
10. T. Berners-Lee, R. T. Fielding, and H. Frystyk Nielsen, "Hypertext Transfer Protocol—HTTP/1.0," *HTTP Working Group Internet-Draft, http://www.w3.org/hypertext/WWW/Protocols/Overview.html*, March 1995.
11. "Coding of Moving Pictures and Associated Audio—for Digital Storage Media at Up to About 1.5 Mbit/s," *ISO Standard IS 11172*, ISO Central Secretariat, 1, rue de Varembé, Case postale 56, CH-1211 Genève 20, Switzerland, November 1992.
12. "Video Coding for Low Bit-Rate Communication," *ITU-T Recommendation H.263*, International Telecommunication Union, Place des Nations, CH12-11 Geneva 20, Switzerland, May 1996.
13. T. Berners-Lee and D. Connolly, "HyperText Markup Language Specification-2.0," Work in Progress (*draft-ietf-html-spec-01.txt*), CERN, Hal Computer Systems, IETF Secretariat, c/o Corporation for National Research Initiatives, 1895 Preston White Drive, Suite 100, Reston, VA 20191, February 1995.
14. W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression*, Van Nostrand Reinhold, New York, 1992.
15. W. R. Cheswick and S. M. Bellovin, *Firewalls and Internet Security*, Addison-Wesley Publishing Co., Inc., Reading, MA, 1994.
16. "Terminal for Low Bitrate Multimedia Communications," *ITU-T Recommendation H.324*, International Telecommunication Union, Place des Nations, CH12-11 Geneva 20, Switzerland, 1995.
17. "Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 Kbit/s," *ITU-T Recommendation G.723*, International Telecommunication Union, Place des Nations, CH12-11 Geneva 20, Switzerland, 1996.
18. "Video CODEC for Audiovisual Services at p × 64 Kbit/s," *ITU-T Recommendation H.261*, International Telecommunication Union, Place des Nations, CH12-11 Geneva 20, Switzerland, July 1990.
19. S. S.-P. Chang, J. J.-C. Chen, E. Feig, M.-H. Lin, L. K. Liu, and J. H. Morgan, "IBM's H.263 VideoCodec," presented at the High Definition Media Technology and Applications Workshop, Taipei, Taiwan, October 14–16, 1996.
20. L. K. Liu and E. Feig, "A Block-Based Gradient Descent Search Algorithm for Block Motion Estimation in Video Coding," *IEEE Trans. Circuits & Syst. for Video Technol.* **6,** No. 4, 1–6 (August 1996).
21. E. Feig, "A Fast Scaled DCT Algorithm," *Image Processing Algorithms and Techniques, Proc. SPIE* **1244,** 2–13 (June 1990).
22. E. Feig and S. Winograd, "Fast Algorithms for the Discrete Cosine Transform," *IEEE Trans. Signal Processing* **40,** No. 9, 2174–2193 (September 1992).
23. D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1992.
24. M. Willebeek-LeMair, Z. Y. Shae, and Y. C. Chang, "Robust H.263 Video Coding for Transmission Over the Internet," *Research Report RC-20532*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1996.

Received November 6, 1996; accepted for publication June 5, 1997

Marc H. Willebeek-LeMair *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (mwlm@watson.ibm.com).* Dr. Willebeek-LeMair received the B.S. degree in computer and electrical engineering from George Mason University, Fairfax, Virginia, in 1985, and M.S. and Ph.D. degrees from the School of Electrical Engineering at Cornell University, Ithaca, New York, in 1988 and 1990, respectively. He is currently managing the Multimedia Networking group at the IBM Thomas J. Watson Research Center, specializing in the development of networked multimedia systems. Dr. Willebeek-LeMair joined IBM in 1990 as a Research Staff Member in the Research Division High Bandwidth Systems Laboratory. His research interests include real-time networked applications such as desktop videoconferencing and audio/video streaming, high-bandwidth communications, computer architecture, parallel processing, and interconnection networks. Dr. Willebeek-LeMair is a member of the IEEE Computer Society, the IEEE Communications Society, Alpha Xi, and Eta Kappa Nu.

Keeranoor G. Kumar *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (kumar@watson.ibm.com).* Dr. Kumar received the B.Sc. degree in physics in 1978 from Kerala University, India. He received the B.E. degree in electronics and communication in 1981 from the Indian Institute of Science, Bangalore, and the Ph.D. degree in computer science in 1989 from the Indian Institute of Technology, Bombay. Dr. Kumar's research contributions have been in the areas of real-time distributed computing, models of parallel programming, and compiling for parallelism. His current research interests are in the area of system and network architectures for multimedia.

Ed C. Snible *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (snible@us.ibm.com).* Mr. Snible is a Software Engineer in the Multimedia Networking group, Internet Department, at the IBM Thomas J. Watson Research Center. He received his B.S. in computer engineering from Arizona State University in 1990, joining the Research Division in 1997. At IBM, Mr. Snible has implemented Bamba components in Plug-in, ActiveX, and IBM MediaBeans (Java) frameworks. Current research interests include multimedia authoring, design patterns, and working with user interfaces.