Large Scale Experiments on Low Bit Rate Multimedia Broadcast

Z.Shae*, X.Wang, and S.P.Wood

IBM T. J. Watson Research Center
30 Sawmill River Rd, Hawthorne, NY. 10532 USA

ABSTRACT

This paper contains our experience with low bit rate multimedia streaming and broadcast, as applied to the Internet/Intranet, and focuses on two of the enabling technologies: 100% Java clients and broadcast reflectors. Interpreted Java is slower than compiled C/C++ and Java platforms do not currently support video and audio synchronization. Various techniques to improve Java performance and to reduce code size are provided in detail. A novel video and audio synchronization mechanism for the pure Java environment is devised and investigated. This paper also describes a hierarchical reflector network architecture which, superimposed on the Internet, is a practical alternative for broadcasting of events to massive client audiences when multicast support of such audiences in the current Internet is questionable and remains untested.

Keywords: Multimedia Broadcast, Reflector, Java Optimization, Streaming, Audio/Video Synchronization

1. INTRODUCTION

Multimedia streaming and broadcast across the web is exploding. Improved processor performance, higher bandwidth connectivity and improved compression are important technology trends which are fueling this onslaught. Our experiments in this area evolved from initial investigations into key technologies that would allow the integration and composition of multimedia with Internet HTML pages and permit media-rich web-based content to be developed. Applications, such as distance education and business seminars, based on the playback of stored media were envisioned. Broadcasting, of either live or recorded content, extended the application potential to include transmission of live speeches and presentations.

Bamba, developed at the T.J. Watson Research Center, is a solution for low-bit rate audio/video broadcasting and streaming across the Internet. Bamba uses standard compliant codecs that can compress video and audio clips down to low-bit rates compatible with 28.8k modems and can also scale up, to provide higher quality video, to rates in the low hundreds of kilobits per second. The compressed audio and video data is packetized using the Bamba stream format that was designed to operate in a UDP/IP environment where packets can be lost at any time during the transmission. It was also designed to allow new clients to join a multicast transmission at any point during an ongoing broadcast. The Bamba stream format allows the server to fragment video and audio at arbitrary data boundaries that can significantly increase server efficiency. Bamba also includes patented provisions for fast video recovery from network error [1,2]. Bamba had its public debut when it was featured as a technology demonstration on the official web site of the 1996 Olympics and has then been made available for download from the IBM alphaWorks web site.

The initial Bamba client player was designed as a browser plug-in. With the advent of Java and the rapid advances of the technology in its application to the Internet, we decided to implement a pure Java version of the Bamba streaming player that would allow us to overcome both platform dependency and the complexity of installation for the average user. The move to the Java environment, with its notoriously slow code execution and lack of sufficient multimedia support, was extremely challenging given our goals: small code size, fast speed, and high quality multimedia performance.

Reflectors, the other technology to be discussed here, were developed as a practical alternative to multicasting. The deployment of DVMRP[3] multicast algorithm on the Internet would cause periodic network flooding due to its broadcasting

* Correspondence: Z.Shae. Other author information: X.Wang; Email: xiping@us.ibm.com; S.P.Wood; Email: woodsp@us.ibm.com; Z.Shae; Email:zshae@us.ibm.com;

Part of the IS&T/SPIE Conference on Visual Communications and Image Processing '99 • San Jose, California • January 1999
SPIE Vol. 3653 • 0277-786X/98/$10.00

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

in the tree pruning process. Although various multicast algorithms [4-7] have been proposed or implemented, the multicast support of massive audiences in the current Internet is still questionable and remains untested. Therefore, to support broadcasting such events, we have developed a reflector solution that provides efficient network distribution of the massive live multimedia streams. Some time later, Real Network Inc. installed a similar reflector network (RBN [13]) maintained by MCI for its Realplayer broadcasting. We also mention briefly in this paper some reflector network applications beyond their primary use in stream replication and distribution.

This paper now presents the details of our efforts in these two major enabling technologies developed for Bamba. First a brief overview of the Bamba architecture is covered in section 2. Section 3 details the reflector network as a solution to the problem of efficient distribution of live broadcast streams. Section 4 covers the pure Java player, and contains an overview of the issues followed by the techniques that used to optimize the Java code and to synchronize video/audio playback. At a time when many people were still doubtful that pure Java could be used to implement a multimedia player we show results that demonstrate satisfactory end-user performance. Finally, a summary is given in section 5.

2. OVERVIEW OF BAMBA ARCHITECTURE

An introduction to Bamba and architecture overview has been published before [9]. However a short description of essential background material for the topics discussed in this paper is included here. The Bamba player is designed as a client-side extension to a standard Internet browser, plug-in or applet, that works with an unmodified HTTP web server to stream stored multimedia content to a client. This is shown in figure 1 below.

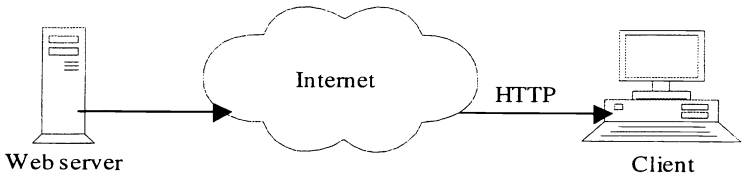

Figure 1. Streaming multimedia from a web server to a client.

The multimedia content is played while it is being received – so called "streaming". The only server side requirement is to specify a new MIME type for the Bamba multimedia content.

The Bamba live broadcast client player is also launched from a web page. A parameter, authored in the plug-in's HTML <EMBED> tag or <APPLET> tag, now specifies another IP address from which to receive the live multimedia stream. The live source is captured and encoded at a transmission station and fed to a reflector network. The client connects to one of the reflectors in the network, as identified by the IP address authored in the HTML tag parameter, and joins the live broadcast. This arrangement is shown in figure 2 below; only one reflector has been shown for clarity.

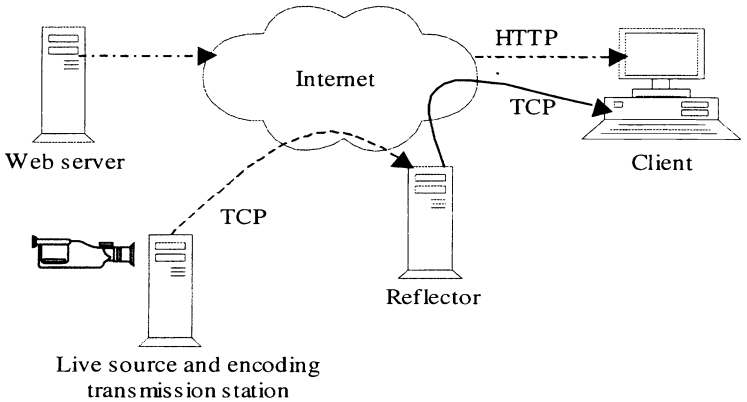

Figure 2. Broadcasting live multimedia to a client via a reflector.

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

For live broadcasts there is a security issue with Java as applets are only allowed to make connections to the host from where the applet byte code originated. So for Java clients we also run a minimal web server on the reflector machine whose sole purpose is to supply the applet code. The HTML page containing the applet is still obtained from the usual web server to ensure that the reflector fulfills its primary role of distributing media.

3. REFLECTORS

There are some major differences between broadcasting a live stream and streaming stored content to a web browser. The latter can be done using only a standard HTTP server - a key goal for Bamba was to provide a low cost entry level into multimedia streaming and encourage web-based multimedia application experiments. For stored content each client requires a unique media stream from the source. Any other clients that may be viewing the same stored content will not, in general, be viewing at the same position in the media and also may want to exert individual control over their stream to pause, resume, rewind etc. This is not the case for live broadcast.

With live streams all clients will be viewing the same live content at the same time and at the same position in the media stream. With many clients viewing the same media stream simultaneously efficient distribution is required to minimize network congestion. Multicast, for IP networks, is one solution that has been developed for efficient distribution. Drawbacks to using this approach are the current sparseness of deployment and problems in traversing firewalls. So for live Bamba we chose to use TCP connections that would allow us to easily traverse firewalls and manage the distribution problem by deploying hierarchical reflectors in the network. A single reflector may support many clients and takes but a single live stream as input and rebroadcasts it to the many clients connected downstream. However, even for a live broadcast using this approach, each client, at some point in the network, requires an individual feed to their end station. Through appropriate hierarchical allocation of reflectors live streams can be widely distributed and allow clients to connect to a local reflector to minimize the path length of those individual feeds. Reflectors can be cascaded and appear as just another client to an upstream reflector (see figure 3).

A live transmitter was designed to capture audio/video and compress the streams in real-time as a source for the live broadcast. The stored content client player was modified but remained an extension to a web browser. The player would now make its own separate connection to the live feed, as designated by a parameter which gave the IP address for the live feed either as part of the EMBED or APPLET tag for the plug-in and Java applet respectively. With a network of reflectors this parameter would be set for the optimum reflector nearest the client.

3.1 Technical details

In the live transmission station audio and video are converted from analog inputs to digital form, compressed and then packetized into the Bamba stream format. The packets are then transmitted over a TCP connection that is established by the reflector to the transmission station. The reflector manages multiple connections to downstream clients, where clients can be other reflectors or actual end-users. Each of these connections is setup by the clients who establish a direct TCP connection to the reflector given its IP address. Using TCP/IP allows the connections to easily traverse firewalls and maintain high quality.

Clients may join an ongoing broadcast at any point in the live transmission. The first transmission that the reflector makes to the client is the Bamba control header that was broadcast from the live transmitter at the start of the stream. The reflector stores this header and sends a copy to the client immediately after the connection is initiated and before sending the live stream. To broadcast the live stream the reflector maintains a list of connected clients and replicates every live stream packet received from its upstream source to each client that is currently connected downstream.

Multiple broadcast channels can share the same physical reflector node. The reflector node's resources are consumed based on demand but upper limits can be set for the number of connections per broadcast channel as well as for the total number of connections per reflector. Reflectors can be cascaded and hence the reflector network can be scaled to meet demand. A reflector may also be configured for multicast when connected to multicast enabled networks. For example, point-to-point TCP connections may be established between reflectors through firewall boundaries that separate Intranets from the global Internet. Within the Intranets the reflector can distribute the broadcasts via multicast connections.

The reflector concept also provides a platform in which custom features can be added. For example streams can be transcoded from higher to lower bit-rates to accommodate different network bandwidths or client capabilities. It is also

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

possible to have different audio tracks, for the same video feed, and route these audio tracks to different reflectors depending on the reflector's local audience preferences.

A hierarchical Bamba reflector configuration is illustrated in figure 3 below. In this example a live broadcast station is transmitting to a 'root' reflector in the Internet which in turn is forwarding the signal to multiple 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 as needed. The lines with arrowheads represent TCP connections and the direction in which the connection was made from initiator (connect) to target (listen).

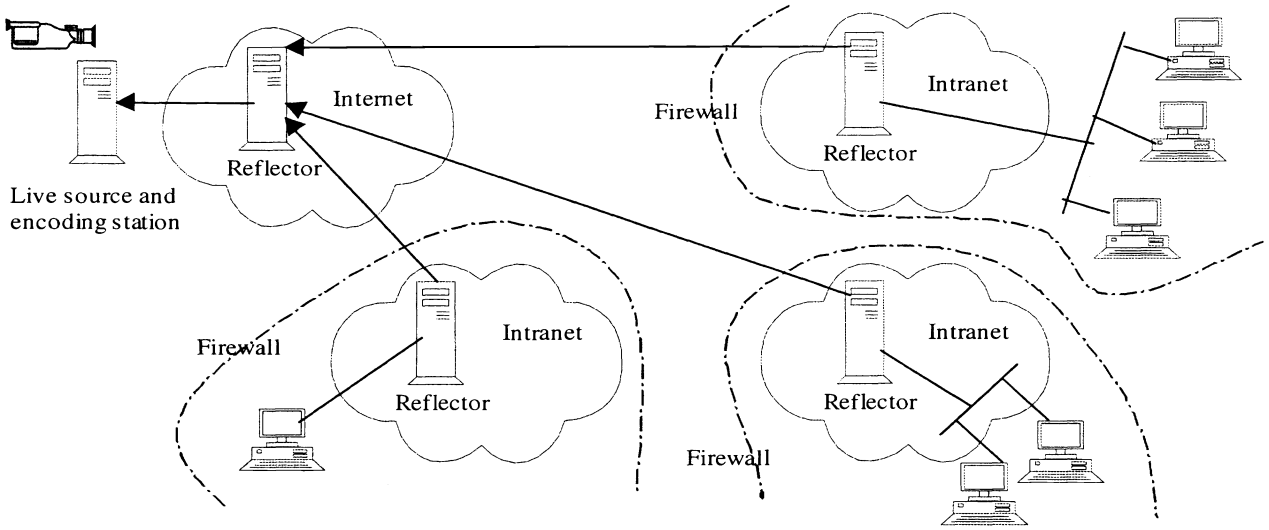

Figure 3. Hierarchical reflector network example.

Using TCP connections, to send these live streams, has advantages from the perspective of firewalls. However it causes additional work for the reflector since any attempt to send to a congested downstream connection can block. To ensure the live stream is delivered to all other clients this blocking condition is detected and no data is replicated for any congested client until the blocking condition disappears. So as far as each client is concerned it must deal with the stream as though it has unreliable delivery even though all the connections from the live source are reliable TCP connections. The Bamba stream format and client were however designed to run over UDP and can deal with such loss. Indeed joining the live stream at an arbitrary point is the case where all the live feed so far is lost - with the exception of the stream header that the reflector stores and forwards to ensure the client always gets an initial control packet.

3.2 Reflector network experiment

The Bamba live broadcast system was designed to support the transmission of live audio and video to multiple recipients simultaneously across the web. A key component to the system is the reflector which provides efficient distribution and scalability for the system. It has been operational since June of 1996 and has been used to transmit live audio feeds to audiences of sizes into the 100000s for events such as the 1996 Olympics, Wimbledon, the Deep Blue Chess matches, and even live concerts in Brazil.

The reflector used for the experiments establishes a TCP connection for each feed, both upstream to the source and downstream to each connected client. TCP was chosen to allow the reflectors to traverse common firewalls. Thus the network of reflectors was superimposed over the existing IP network - it would however be possible to set up dedicated bandwidth for each reflector's live input feed to minimize any disruption to that feed. For our experiments the input feed was susceptible to network problems, but by only sending one feed over the long distance we were minimizing any disruption.

For example, a reflector network was setup worldwide to allow IBM employees to listen to various events. The IBM CEO's broadcast to the IBM employees was one such event. Reflectors were set up both in the USA (Southbury, Connecticut;

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

Raleigh, North Carolina; Austin, Texas), in Canada (Vancouver, BC), and worldwide (Belgium, Croatia, Denmark, France, Germany, Hungary, Ireland, Italy, The Netherlands, Norway, Russia, Slovenia, South Africa, Spain, Sweden and UK). There were two live encoding stations, one in the USA in Southbury and one in Europe in Belgium. As the broadcast was also being televised and transmitted via satellite a local encoding station was also setup in Europe. Employees came to a central web-site in IBM and then were directed to choose the nearest reflector. Broadcasts done this way were well received with employees worldwide some of whom could now listen to the live audio broadcast where otherwise they might not have been able to.

The above describes a reflector network that was both manually configured and administered and to which clients made a manual decision as to which reflector was best to connect to. Further experiments were done which permitted centralized management over a network of reflectors. Where, more than one live feed was permitted, reflectors could be brought on-line as required and reflector connectivity dynamically altered to optimize the reflector network. To which reflector a new client should connect was automated for optimal performance with reflector loading being distributed, managed and adjusted.

4. JAVA STREAMING PLAYER

The Java streaming player consists of a network interface, demultiplexer, audio decoder/renderer, video decoder/renderer and Java applet controller as shown in figure 4 below. The applet plays exactly the same low-bit rate Bamba encoded content that the native plug-in plays. The Java player can also handle the streaming playback of both stored content and live broadcasts.

When playing stored content the Java applet (streaming player) and media content are loaded onto an HTTP server. When the client navigates to the HTML page containing the applet, the Java streaming player code is fetched by the Java enabled browser, using the HTTP protocol, and executed locally. The Java applet itself then opens an HTTP (or TCP) input stream to play the media content. The Java applet controller implements a VCR-like user interface to allow the user control over the playback. When playing live content the arrangement is little different and is described in the reflector section above.

The implementation of the pure Java streaming player was extremely challenging to achieve acceptable multimedia playback performance and at the same time kept the code small to minimize applet download time. So the main effort, having ported our native code to Java, was on performance improvement, code size reduction and audio-video synchronization.

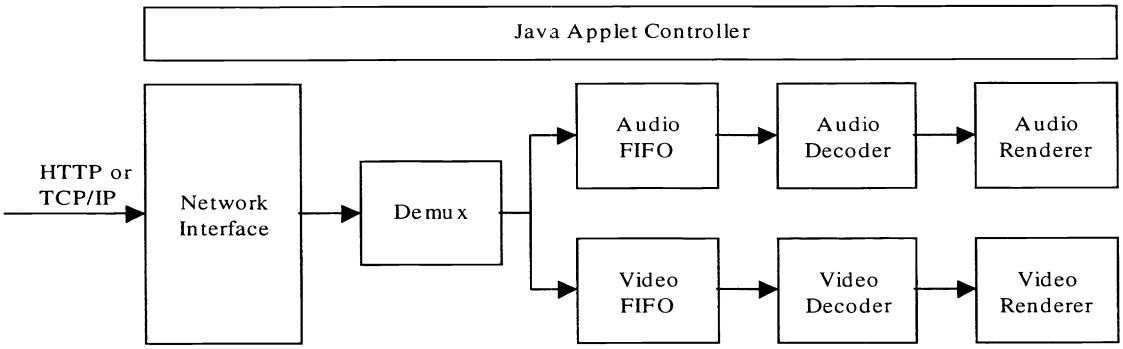

Figure 4. Bamba streaming player Java applet architecture

4.1 Performance optimization

The audio and video compression standards used by Bamba are designed for very low bit-rate multimedia applications. The respective decoders contain complicated algorithms that take a significant amount of processing to decompress the data. Java is still much slower than the natively compiled C/C++ that we used to code the Bamba plug-ins. Even when the Java VM contains just-in-time (JIT) compilation technologies the performance is still slower than C/C++. To achieve an acceptable performance for audio/video playback in pure Java the code had to be analyzed and optimized for speed and code size. This code optimization was undertaken in three main aspects:

Algorithmic : Use faster alternative algorithms to speed up process
Code size : Use alternative coding methods to minimize code size
Code speed : Use better programming techniques to maximize performance.

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.