Title
Remote control for videoconferencing

Author
Perry, Marcia

Publication Date
2000

LBNL-44352

Remote Control for Videoconferencing

Marcia Perry, Deborah Agarwal

Information and Computing Sciences Division
Ernest Orlando Lawrence Berkeley National Laboratory
Berkeley, CA 94720

November 1999

This work was supported by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division under U.S. Department of Energy Contract No. DE-AC03-76SF00098.

Remote Control for Videoconferencing*

Marcia Perry and Deborah Agarwal

Information and Computing Sciences Division
Ernest Orlando Lawrence Berkeley National Laboratory
One Cyclotron Road
Berkeley, California 94720
MPerry@lbl.gov, DAAgarwal@lbl.gov
Telephone: (510) 486-6786   Fax: (510) 486-6363

ABSTRACT

We have designed, implemented, and deployed a camera control system and a conference controller that provide remote control capabilities for videoconferencing over the Internet. The camera control system allows users to pan, tilt, and zoom the cameras, switch between cameras, and get a picture-in-picture view from their desktops. The conference controller allows conference participants to not only start and stop the media tools on a remote host, but also to dynamically change settings and turn transmission on and off. It supports the *vic* (video) and *vat* (audio) Internet videoconferencing tools and enhances their usability by providing an integrated and secure user interface for local and remote control of these applications. This paper describes the design and implementation of the camera control system (*devserv* and *camclnt*) and the conference controller (*confcntlr*). The remote control capabilities offered by these tools have changed the videoconferencing paradigm to one of telepresence. With these tools remote users can "walk" around the room, focus in on objects, and actively participate rather than just observe.

1. INTRODUCTION

The implementation of IP multicast over the Internet has inspired videoconferencing tools for video, audio, session directory, conference management, and shared workspace applications. These tools are built as standalone applications and integrated videoconferencing systems. However, they are designed for people sitting directly at the computer terminal participating in a videoconference and in some conferencing situations there may not be anyone to sit at the computer at a participating site. In the case of collaboratories, our experience has been that the researchers present at an experiment site do not want to tend to the videoconferencing tools in order to select views for the remote collaborators. This is also the case for participants in conference room meetings. However, if no one is at the sending host to execute a video tool or does not turn on transmission, remote users have no way of receiving an image. Also, if the person watching the video wishes to move a remote camera or change the remote settings and is unable to do so, it is frustrating to that remote participant. Remote control of videoconferencing

*This work was supported by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division under U. S. Department of Energy Contract No. DE-AC03-76SF00098 with the University of California.

devices can provide a non-disruptive means of moving cameras and improving audio quality locally.

As part of the Distributed Collaboratories project of the Imaging and Distributed Collaborations Group at Lawrence Berkeley Laboratory, we have designed, implemented, and deployed a camera control system and a conference controller. These tools give the remote user a sense of telepresence by providing remote control capabilities for videoconferences over the Internet. With the remote camera control and conference controller, collaborators can "walk" around a remote room, focusing in on what is taking place. This capability allows users to feel more like participants instead of observers. The camera control system consists of a server (*devserv*) and a client (*camclnt*) to drive serial-controllable video devices. *Devserv* is run on the machine directly connected via serial ports to the camera system. *Camclnt* is the user interface that can be run remotely or locally to control the cameras. Through the *camclnt* interface the user can control camera pan, tilt, zoom, and picture-in-picture. The *devserv* and *camclnt* programs communicate via IP multicast and UDP unicast.

The conference control tool, *confcntlr*, enhances the usability of the media tools *vic* (video) and *vat* (audio) by providing an integrated and significantly enhanced user interface to these tools. *Confcntlr* allows conference participants from local and remote sites to change media tool settings. *Confcntlr* is based on a peer-to-peer architecture and it uses TCP connections to exchange messages over the Internet and IP multicast for communication with the media tools on the local host.

The remainder of this paper is organized as follows. Section 2 surveys related work in multicast-based videoconferencing. Section 3 discusses the remote camera control system. Section 4 discusses the conference control tool. Section 5 summarizes the paper and suggests future work.

2. RELATED WORK

Many of the early public domain, IP multicast-based videoconferencing tools were single media standalone applications such as the *sdr* session directory[4], and the mbone tools--*vic* and *vat* for video and audio and *wb*, a shared whiteboard[5]. Later development involved enhancing existing tools and building integrated systems. For example, support for new video cards was added to *vic* and the Robust Audio Tool (*rat*), which offers improved audio quality, was developed[10]. Although v*ic*, *vat*, *wb*, and *rat* are independent applications, they all have the ability to use a local multicast-based "conference bus" or "message bus" for interprocess communication.

An early conference management tool, the MultiMedia Conference Control program (*MMCC*), provides an integrated user interface to media tools, and offers session creation and invitation capabilities. Its "autopilot mode" allows users to accept invitations automatically, so the media tools can be started from a remote host, but settings cannot be changed remotely while the tools are executing[13]. More recent integrated conference management systems include the Multimedia Internet Terminal (*MINT*)[14], *mStar*[8], the MASH project[7], and the CORE2000 Collaboration Environment[9]. MASH has added to *vic* and *vat* the *collaborator* application, which provides an integrated user interface to the media tools[12]. MASH has also implemented a remote-controlled version of vic (*rvic*) to allow conference participants in one room (without a technician) to manipulate a shared video display. An *rvic* server displays a set of windows representing the various video sources and supports different window layouts [6]. In addition to applications, MASH provides a network and media toolkit from which new applications can be built, and this toolkit includes agents for driving some serial devices. MASH is a research tool and so the priority is not on robustness or completeness of the tools.

mStar is a commercial product and includes a controller that drives a Canon VC-C1 camera and an Extron 100 videoswitcher. Its development framework defines mechanisms for remotely controlling tools and parameters (e.g., stopping tools or changing bandwidth from a remote host). This control is intended for administrators rather than for conference participants. CORE2000 provides remote startup and termination of applications and a camera controller for pan, tilt, and zoom. When a user starts a tool, it is automatically launched on all participants' hosts and, when a user terminates a tool, he or she is asked whether to stop the tool for everyone. Once the tool is executing its settings cannot be changed by a remote host. CORE2000 supports third-party applications (e.g., CuSeeMe, Televiewer) and provides a framework for porting new tools to its Java environment.

The above systems provide cross-platform, integrated user interfaces for establishing, joining, and controlling multimedia conferences from the desktop. Although features vary from system to system, they support a wide range of collaborative capabilities including invitation, voting, floor control, chat, media archiving and playback, resource reservation, and conferencing via a web browser. And while several of the above tools provide some remote control capabilities, they do not offer all of the features needed by a distributed collaboratory. For example, these tools lack the abilities to remotely drive a variety of cameras and to change the configuration of media tools during execution. Since these capabilities are important for telepresence, we developed the remote camera control system and conference controller to directly address these needs.

3. REMOTE CAMERA CONTROL SYSTEM

The camera control system consists of a device server (*devserv*) and a client (*camclnt*) which together allow users to control the video devices. *Devserv* is run on the host connected to the devices and *camclnt* is the graphical user interface that can be run anywhere. Participants using *camclnt* can select the camera to view and can pan, tilt, or zoom any available camera. They can also create or move a picture-in-picture view if there is a videoswitcher. In addition to providing a means of controlling video devices from the desktop, *devserv* and *camclnt* are extensible and cross-platform. *Devserv* is written in C++ and *camclnt* is written in Java and both have been tested on Solaris, FreeBSD, Linux, Irix, and Windows95/98/NT. The server currently supports the Sony EVI-D30/D31 and Canon VC-C1/VC-C3 cameras and the Panasonic WJ-MX50 videoswitcher.

3.1 System Design

Client requests are transmitted using UDP unicast and IP multicast connections; the server uses IP multicast to send messages. Servers and clients can execute on the same or different hosts and any number of hosts can join a multicast group. Our system supports both the socket interface for network communication and the common communication library developed under the Collaboratory Interoperability Framework (CIF) project[1]. The CIF library provides a simple uniform interface to low-level network protocols providing reliable and unreliable unicast and multicast. CIF has implementations available in both C++ and Java and these implementations interoperate seamlessly.

Requests to move the video devices are sent by the client to a server using UDP unicast. The server then drives the devices (via RS232 communication) and multicasts the resulting status. Clients can also send a request to the server to send a description message. Commands and descriptions are ASCII strings, defined in our Remote Camera Control Language[2]. All messages contain a header with a timestamp and the server's name. Descriptions contain status information such as conference information (address, port, etc.) and the state of the connected devices (e.g., type, degrees of freedom, current positions). Commands specify the device, degree of freedom, and the appropriate values and allow absolute, relative, and fractional camera movements. For example, "cam 3 tilt R -20 1" is a command for camera three to tilt minus twenty degrees relative to the current position at the maximum speed, and "cam 1 pan F 0.5 1" is a command to pan camera one by one-half of an image to the right, at maximum speed. Other clients may be written to work with *devserv* by implementing the Remote Camera Control Language.

In order to control access to the devices, we have incorporated the Secure Socket Layer (SSL) to provide a secure connection between the client and the server, and the Akenti authorization system to verify client authorization[15]. Servers and clients are identified by X.509 identity certificates.

3.2 Devserv

Devserv's class structures for networking and devices are shown in Figure 1. In the network hierarchy, classes for specific connection types (UDP unicast, IP multicast, and SSL) are derived from a Network abstract base class that encapsulates common socket properties (e.g., identifier, open/close, send/receive). For CIF, objects from the CIF library are also used. The hardware device classes mirror physical objects. A class for each device is derived from the class that encapsulates the properties of its category (e.g., the Camera or Videoswitcher class). These base classes are derived from the Device abstract base class which represents attributes that are common to all programmable devices (e.g., move, transmit, receive). At runtime an object is instantiated for each device that is connected. Classes for new devices can be easily derived from the existing classes (e.g., a new videoswitcher could be derived from the Videoswitcher class). The network class structure can be extended similarly.

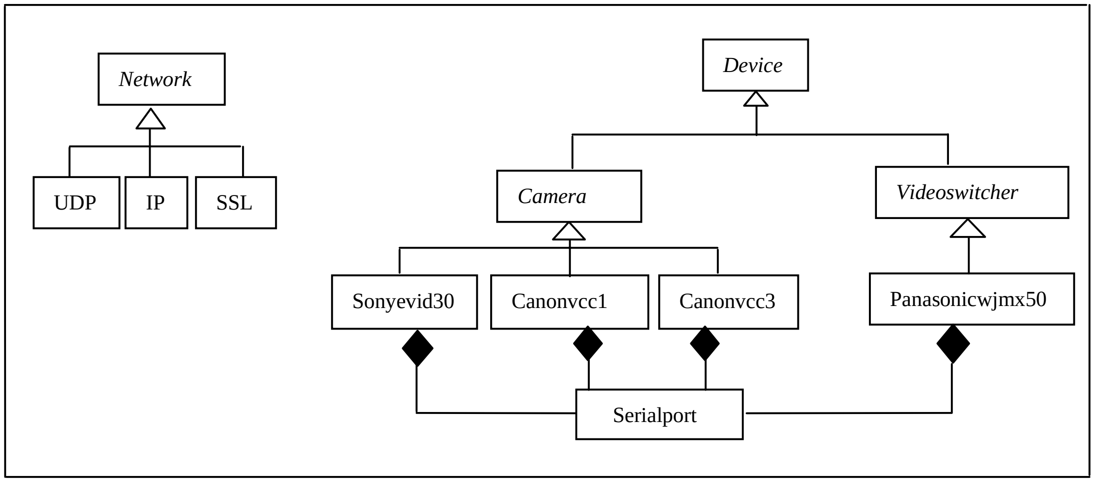

Figure 1: Devserv's Major Class Structures

Upon startup, *devserv* determines the hardware and network configuration for the host from a configuration file and then opens the serial port and network connections and initializes the devices according to the configuration. It then receives and processes requests until program termination. For security, the Akenti software provides authorization for each identity so that *devserv* can make access control decisions. If security is enabled, an SSL connection is established from *camclnt* to *devserv* to exchange and validate identity certificates. The shared secret that is generated by the SSL handshake (the master-secret) is cached by *devserv* and used to make access control decisions when requests arrive. *Devserv* multicasts descriptions periodically or after carrying out a request. Requests are sequenced by their timestamps.

The *devserv* program is threaded: a thread is created for each device and network connection. The device threads initialize, run, and shut down the devices. The threads for network connections process incoming requests while the main thread sends descriptions. Thread synchronization uses "wait-and-signal" mechanisms and synchronize serial ports by locking objects.

3.3 Camclnt

Camclnt uses the Java I/O and networking packages to implement communication. For CIF, objects from the CIF Java implementation are used. At startup *camclnt* opens its network connections and then multicasts a request for a description message. The descriptions allow *camclnt* to discover the addresses of hosts running servers and the devices connected to each server. This information is cached by *camclnt*. *Camclnt*'s graphical user interface displays a list of the servers discovered. When a server is selected, the window is reconfigured to show the information for that server.

Camclnt's graphical user interface is based on Java's Abstract Window Toolkit (AWT). The Java Media Framework (JMF) tools are incorporated to display the video. The main window contains controls for selecting a server, selecting cameras to view and move, and specifying device commands. Commands are sent to the server when the user selects a device, clicks in the pan-and-tilt area, or manipulates a zoom control. For security, each command also includes a signed hash of the command. Fractional moves and a picture-in-picture view are created by clicking and dragging in the camera view area or the JMF player window. The pan/tilt area allows the user access to the entire pan and tilt range of the camera. Figure 2 shows the main window on the right and the JMF player window on the left.

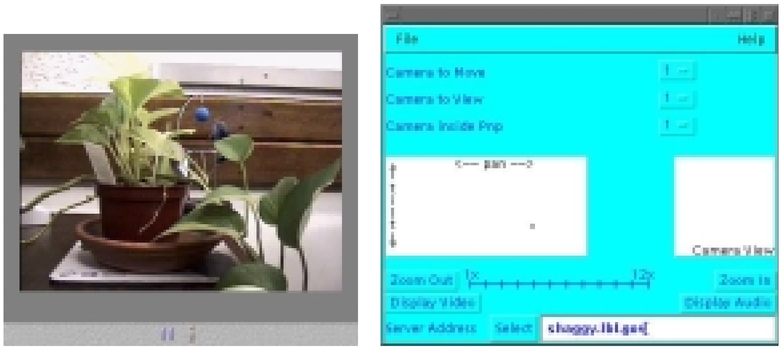

Figure 2: JMF Video Player and Camclnt Main Window

Because servers can send descriptions at any time, *camclnt* uses a main thread to respond to user-triggered events and a separate thread for receiving messages from the server. When a description arrives, the receiving thread updates the configuration information for the server and reconfigures the information in the main window to reflect the server's current configuration. Access to shared objects is synchronized with Java synchronization mechanisms.

4. CONFCNTLR

Confcntlr was developed to control media tools locally and remotely from a unified interface[11]. It supports the following actions:

- start one or all media tools on a local or remote host with user-selected settings
- stop media tools running on a local or remote host
- obtain a remote host's settings and current videoconferencing information
- change settings for a local or remote media tool that is executing
- turn video transmission on or off at a local or remote site

Based on a peer-to-peer architecture, *confcntlr* is meant to execute on each host participating in a videoconference. *Confcntlr* supports encryption to preserve confidentiality and integrity of data exchanged between *confcntlr*s over the public Internet. Security features also allow a user to restrict access to the local *confcntlr*. The security operations are to:

- encrypt/decrypt messages exchanged between conference controllers at remote sites
- allow restricted control to authorized users
- allow users to grant or deny permission for another host to perform an operation unconditionally or on a per-request basis
- provide warnings when changes are made

4.1 Design

The conference controller was designed as a desktop application that permits access to the tools it controls. It is single-threaded, handling all its events within one event loop that utilizes a FIFO event queue. Communication between *confcntlr*s running on different hosts is via TCP connections. Communication between applications running on the same host is via a "conference bus." The conference bus is an IP multicast connection with a time-to-live of zero to restrict messages to the local host. Figure 3 depicts this communication architecture.

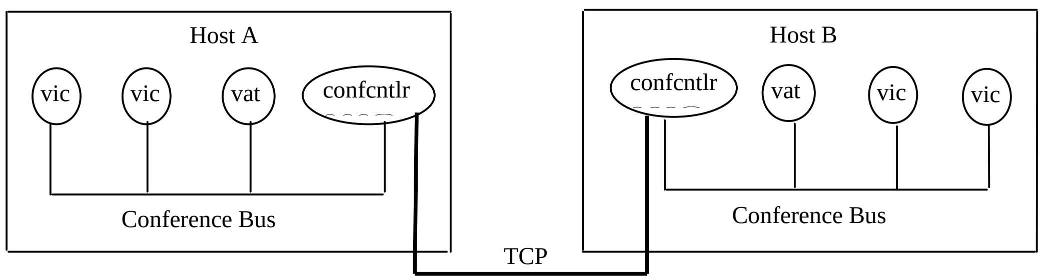

Figure 3: Communication Channels. A TCP connection is for peer-to-peer communication; the conference bus is for interprocess communication.

In order to allow an arbitrary number of remote controllers, a separate TCP connection is used for each request; all other connections are opened once and remain open until program termination. Each conference controller controls one conference session at one site but multiple *confcntlr*s can be executed for simultaneous participation in multiple conferences.

4.2 Implementation

The conference controller was written in Tcl/Tk and C and runs under Solaris, freeBSD, Irix, and Windows. It has been designed as four separate units that work together: a graphical user interface (GUI), a control unit, a network unit, and an encryption unit. Figure 4 shows the relationships of these components.

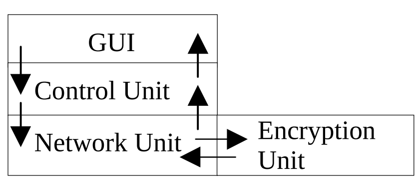

Figure 4: *Confcntlr*'s Components. The arrows indicate the data flow.

All actions go through the control unit. When a user manipulates a GUI control to invoke a local or remote operation, the control unit processes the request. To send a message to a remote host or to a videoconference tool on the same host, the control unit invokes the network component. The network unit invokes the encryption unit when messages are sent to or received from a remote host. The encryption unit uses the SSLeay library to encrypt plain text and decrypt ciphertext with the Data Encryption Standard (DES). When the network unit receives a message, the control unit processes the message and invokes the GUI to display the output. The control unit carries out local operations invoked by the GUI or network unit. To start or stop a media tool on the local host, *confcntlr* spawns or terminates a process for the tool. To change local settings for a tool that is executing, the control unit formats a message and the network unit sends the message to the target tool. When a user invokes a remote operation, the control unit formats a message and the network unit sends it to the remote *confcntlr*. The control unit also formats replies to requests received from other hosts and processes incoming replies.

The network unit is responsible for establishing and closing socket connections and transmitting and receiving all host-to-host and interprocess communication. Requests and replies are usually not sequential and replies are not always sent. There are two host-to-host communication schemes.

1. Host A sends a request to host B. Host A does not wait for a reply unless it is obtaining a remote host's settings. Host B receives the request, processes it, and sends a reply.
2. Host A notifies host B that some event took place (e.g., host A terminated a tool). Host B receives and processes the notification but does not send a reply.

For interprocess communication, each process connected to a multicast channel receives a copy of all messages sent over the channel. If a process recognizes the message type, it processes the message locally and may also forward the message to another conference controller at a remote site. *Confcntlr* sends messages on the conference bus to dynamically change settings on the media tools and processes notifications from the tools that a setting was changed or that it is being terminated.

Confcntlr's graphical user interface is designed to be unobtrusive so that it can run continuously on the user's desktop. It presents a main window with status information and controls for basic operations. Popup windows can be opened for specific categories of functions (e.g., local or remote settings, security, and general conference control). The main window is shown in Figure 5. The status buttons indicate what media tools are running locally and remotely. The image in the lower left displays the remote access level: red, yellow, and green correspond to "allow no one," "allow authorized users," and "allow anyone," respectively.

Figure 5: Confcntlr's main window

The conference window, shown in Figure 6, is used for setting media tool addresses and launching the media tools. Figure 7 shows the window for changing the settings of remote media; the "Local Settings" window is similar. Figure 8 shows the security window for access control, prompts, and warnings options, an access control list, and a "View key" button to invoke an encryption window.

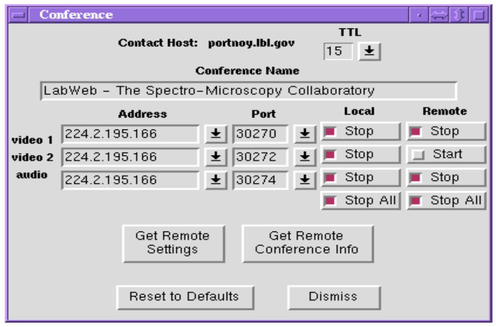

Figure 6: Conference Window

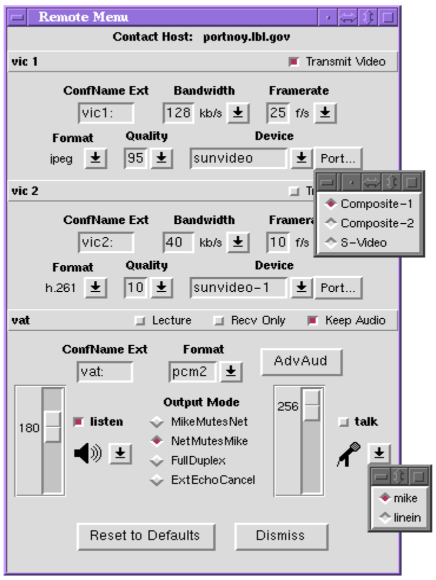

Figure 7: Remote Settings Window

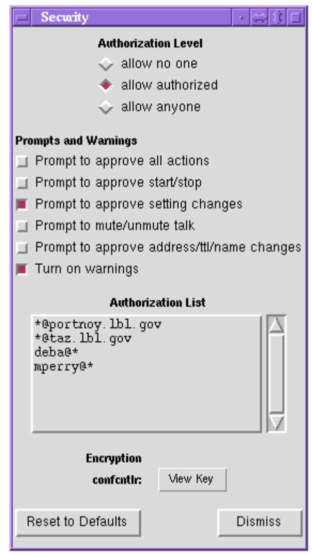

Figure 8: Security window

5. SUMMARY AND FUTURE WORK

This paper describes the design and implementation of a camera control system and a conference controller that together provide remote control capabilities to Internet-based multimedia conferencing. These applications allow users to control the video devices and media tools used in a videoconference from anywhere over the Internet. With these tools the person who is watching the transmission and cares most about how it is received can control the transmission. This new remote control capability has changed the videoconferencing paradigm to one of telepresence in which remote users become active participants rather than passive observers.

The camera control system and conference controller have served an important role in videoconferencing at Lawrence Berkeley Laboratory and have been used in large conference rooms and by the Spectro-Microscopy Collaboratory[3] for videoconferencing. With these remote control tools, participants at the local site can spend their time on scientific experimentation or attending the meeting rather than 'babysitting' the videoconference tools. By presenting a unified interface to the media tools, *confcntlr* has made it easier to manage these separate tools. *Confcntlr* has also been used to remotely instruct a user in operating the media tools. Access control mechanisms allow protection of computer and network resources and have reduced concerns about being watched or heard without a user's knowledge and consent.

Our experience has been that the usability of multimedia conferencing tools is enhanced when there is a unified, intuitive, and configurable interface. Our next goal will be to expand the integrated interface concept into a wide range of videoconferencing capabilities. We plan to offer an expanded implementation that works with a wider variety of media tools. In addition to remote control capabilities, the enhanced system will include mechanisms for floor control, indicating users' availability, and meeting with both groups and individuals. The remote camera control system was demonstrated at the High Performance Distributed Computing Conference (HPDC'98) in Chicago, Illinois, July 1998 and at the SuperComputing Conference (SC'98) in Orlando, Florida, November 1998. *Confcntlr* was demonstrated at the SuperComputing Conference (SC'97) in San Jose, California, November 1997. More information about the project is available at http://www-itg.lbl.gov and http://www-itg.lbl.gov/mbone.

REFERENCES

1. Agarwal, D., et al., *The Collaboratory Interoperability Framework Common Application Programming Interface*, http://www-itg.lbl.gov/CIF/Reports/GcommonAPI.html, LBNL Report #44357.

2. Agarwal, D., Perry, M., *Camera Remote Control Command Language*, http://www-itg.lbl.gov/devserv/Remcam.txt, LBNL PUB-3149.

3. Agarwal, D., Johnston, W., Perry, M., *The Spectro-Microscopy Collaboratory at the ALS*, http://www-itg.lbl.gov/Collaboratories/ALS.html, LBNL Report #37331.

4. Clarke, L., Sasse, A., "Conceptual Design Reconsidered -- The Case of the Internet Session Directory Tool." *Proceedings of HCI'97*, Bristol, UK, August 1997.

5. McCanne, S., Jacobson, V., "vic: A Flexible Framework for Packet Video." *ACM Multimedia*, November 1995, pp 1-19.

6. Hodes, T., et al., "Shared remote control of a video conferencing application: motivation, design, and implementation." *Proceedings of SPIE Multimedia Computing and Networking*, San Jose, CA, USA, January 1999, pp. 17-28.

7. McCanne, S., et al., "Toward a Common Infrastructure for Multimedia-Networking Middleware", *Proceedings of the 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'97)*, May 1997.

8. Parnes, P., Synnes, K., Schefstrom, D., "A Framework for Management and Control of Distributed Applications using Agents and IP-multicast." *Proceedings of the 18th IEEE INFO-*

COM Conference (INFOCOM'99), 1999.

9. Payne, D., Myers, J., "The EMSL Collaborative Research Environment (CORE) - Collaboration via the World Wide Web." The *IEEE Fifth Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises* (WET ICE '96), June 19-21, 1996, Stanford, California.

10. Perkins, C., et al., "Multicast Audio: The Next Generation." *Proceedings of INET'97*, June 1997, Kuala Lumpur, Malaysia.

11. Perry, M., *Confcntlr: A Videoconference Controller*: Masters' Thesis, San Francisco State University and Lawrence Berkeley National Laboratory, Publication Number LBNL-41154, December 1997.

12. Romer, C., "A Composable Architecture for Scripting Multimedia Network Applications." Masters' Report, University of California, Berkeley, July 1998.

13. Schooler, E., "Case Study: Multimedia Conference Control in a Packet-switched Teleconferencing System." *Journal of Internetworking: Research and Experience*, Volume 4, Number 2, June 1993, pp 99-120.

14. Sisalem, D., Schulzrinne, H., "The Multimedia Internet Terminal." *Journal on Telecommunication Systems*, Volume 9, Number 3, 1998, pp 423-444.

15. Thompson, M., et al., "Certificate-based Access Control for Widely Distributed Resources." *Proceedings of the Eighth USENIX Security Symposium (Security '99)*, Washington, D.C., August 23-26, 1999, pp 215-227.