dc: A Live Webcast Control System*

Tai-Ping Yu, David Wu, Ketan Mayer-Patel, and Lawrence A. Rowe
Berkeley Multimedia Research Center
University of California, Berkeley
{taiping, davidwu, kpatel, larry}@bmrc.berkeley.edu

ABSTRACT

Webcasts can adopt techniques developed by television to obtain higher quality. Television production techniques, however, cannot be integrated into webcast production without considering the fundamental differences in the underlying infrastructure. We have developed a general webcast production model composed of three stages (i.e., *sources*, *broadcast*, and *transmission*) to address these differences. The Director's Console (*dc*) is a live webcast production application based on this model. It utilizes a distributed service architecture, which adapts to varying physical infrastructures and broadcast configurations.

1. INTRODUCTION

The development of IP Multicast, the Mbone Tools [11], and commercial streaming media systems (e.g., Apple QuickTime Streaming, Cisco IP/TV, Microsoft Windows Media, and Real Networks) has lead to a rapid growth in the use of streaming media on the Internet. Several hundred live webcasts are produced each week that are being viewed by tens of thousands of viewers [20]. Webcasts are becoming an alternative to traditional television broadcasts.

The television industry has developed many techniques to produce high quality video programs that can be used to improve webcasts. One technique is to switch between different video sources. In a typical television news production, several video feeds are sent to a central studio. The director selects the stream that best portrays the content and broadcasts that stream to the audience. News broadcasts, for example, introduce a subject at the anchor desk and cut to different sources as needed to present the story (e.g., recorded material, a live-remote report, etc.). Another television technique uses video effects to combine several video sources into one stream (e.g., composition effects), to place text on an image to describe who or what is being displayed (e.g. titling effects), and to transition between streams (e.g., fade effects). An interview show, for example, might display an interviewer in one window and a remote person who is being interviewed in a second window composited side-by-side with titles to identify the program, the participants, or the location of the remote source.

Webcast producers are reluctant to integrate conventional broadcast television techniques into webcasts for several reasons. First, television production equipment is very expensive. Second, the technologies and models used in broadcast television typically require many people to operate the system. Third, most broadcast television programs produce a fixed data rate, single video stream with no interaction. Fourth, webcasting has a fundamentally different production model than television broadcasting and the tools and interfaces used to produce high quality webcasts must reflect the underlying infrastructure.

We have produced the "Berkeley Multimedia, Interfaces and Graphics (MIG) Seminar" webcast on the Internet Mbone since 1995. The seminar features invited lecturers who give formal presentations. The seminar takes place in a room equipped with a variety of audio/visual hardware, which is used to produce the webcast. We have used the Berkeley MIG Seminar webcast as a practical experiment to explore the problems associated with webcast production.

Our webcast is comprised of two video streams and one audio stream. One video stream focuses on the speaker, called the *speaker stream*, and a second stream focuses on the presentation material, called the *content stream*. Figure 1 shows a sample webcast from the remote viewer's perspective. The director can switch any of several video sources into either stream. For example, when a member of the local audience asks a question, the content stream can be switched to a view of the audience so that a remote viewer can follow the conversation. Or, if a remote person asks a question, the content stream can be switched to the remote viewer and the projector in the classroom can be switched from the presentation material to the remote person. These

*This research was supported by NSF grant ANI-9907994, NSF equipment grant CDA-9512332, and contributions from Fujitsu, Intel, and NEC.

examples illustrate the complexity of the webcast control problem. The director must control the production viewed by the students in the classroom and, at the same time, he must control the production viewed by a remote participant.

Figure 1: An example MIG Seminar webcast. The left panel shows a thumbnail for each stream in the session. The right panel displays selected streams with greater resolution and frame rate.

The MIG Seminar originates from a classroom with many audio/video sources, computer systems, and software processes as shown in figure 2. The classroom is equipped with several cameras (e.g., speaker and audience cameras), a VCR, and a document camera. These devices are connected to a conventional video matrix switch.[1] Two capture machines take video from any connected input device. Moreover, they can receive the same input simultaneously. The capture machines digitize the audio and video signals for transmission into two multicast sessions (i.e., one for video and one for audio), which we call the *studio session*.[2]

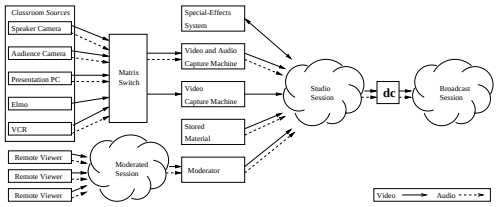

Figure 2: Infrastructure used to produce the MIG Seminar webcasts.

The studio session is analogous to the routing switcher and control room in a conventional television studio. As in television, webcast production sends all sources to one location, the studio session. The producer/director previews all sources and selects a subset of the streams that define the webcast. The selected streams are sent to the broadcast session. Thus, for a viewer to see the slides and the speaker, the producer sets the matrix switch to the correct configuration and broadcasts the two capture machine streams. Video special-effects can be added to the webcast by placing an effects processor on the edge of the studio session. It takes one or more streams from the studio session, renders the effect, and sends the new stream back into the studio session. Stored playback material (e.g., a predefined opening segment) and remote participant video are other examples of sources available in the studio session.

Production of a high quality webcast is a complex operation. In a previous paper, a *Broadcast Manager* application was described that automated the tasks required to initiate a webcast [23]. For example, the MIG Seminar webcast requires that more then ten processes be started on different hosts with appropriate arguments (e.g., multicast addresses and port numbers, media formats, bit-rates, and quality settings).

The Director's Console (*dc*), described in this paper, is designed to control a webcast during live production. It controls the webcast sources, adds effects to these streams, and determines the final output. *Dc* is implemented as a system of distributed processes organized with a service model, which adapts to different physical infrastructure and to different broadcast configurations.

The remainder of this paper presents the design and implementation of *dc*. Section 2 describes the webcast production model. Section 3 describes the system architecture. Next, the user interface is described in Section 4. Section 5 describes the implementation. Section 6 discusses our experiences developing and using the tool and suggests future work. Section 7 describes related research. And lastly, section 8 concludes the paper.

2. WEBCAST PRODUCTION MODEL

A webcast differs from traditional television on several important dimensions. This section describes some of these differences and develops a production model for webcasts. The design and implementation of *dc* reflects this production model. We begin by examining the types and number of data streams that may be involved in a webcast. Next, the source infrastructure used in a webcast is described. Finally, the receivers of a webcast are characterized. In each case, differences between the webcast environment and a conventional television broadcast environment are highlighted and used to motivate developing tools that meet these webcast-specific needs.

Unlike a television broadcast that is limited to one video and audio stream, a webcast can be comprised of multiple video and audio streams along with associated hypertext documents. The number of streams may also be dynamic. For example, during the webcast for the Berkeley MIG seminar, one video stream may be initially used when the speaker is presenting his material and another video stream showing the audience may be added to the webcast during the question and answer period at the end of the seminar. In a traditional television broadcast, switching between a view of the speaker and a view of the audience might be used during the question and answer period. This technique could also be used during a webcast, but providing both streams

[1] A matrix switch is a crossbar switch that can send any input to multiple outputs. A typical switch routes both audio and video signals although the signals can be controlled independently. Matrix switches are also called *routing switches*.

[2] A multicast session contains only one media format. It is specified by a multicast address and port pair.

simultaneously is another possibility because webcasts may have more than one video stream.

The infrastructure available to a webcast is an IP-based data network with cameras and microphones attached to PC's that capture, digitize, possibly compress, and transmit the media streams. A television broadcast, in contrast, requires expensive, special-purpose hardware operated by trained technicians. The sophistication of a webcast production environment may vary from a single video and audio stream to a situation like the Berkeley MIG seminar where multiple video and audio streams, an effects server, remote participants, and auxiliary hypertext streams are used. The tools used to produce a webcast are required to accommodate heterogeneity in the production infrastructure.

The receivers of a webcast are also heterogeneous. For example, in the Berkeley MIG Seminar, a high quality version of the webcast is sent to viewers on the local campus and Internet2 connected sites that have high bandwidth connections and experience little delay. A low quality version of the webcast is sent to Public Internet viewers that may have limited bandwidth connections. The director must manage two different transmissions of the same material. These transmissions may differ in bandwidth and quality and may also differ in the number and type of streams sent. In a television broadcast, only one version of the program is transmitted and the capabilities of the receivers (i.e., television sets) are standardized.

Dc was developed to manage and control the production of live webcasts. It uses a general webcast model composed of three stages: source, webcast, and transmission as shown in figure 3. *Sources* are the streams available in the studio session. From this set of sources, a subset is selected for the *broadcast*. The speaker and content stream, for example, can be chosen from the video sources in the classroom or any source available in the studio session (e.g., speaker camera, audience camera, stored material, or presentation PC). Finally, multiple copies of the webcast, called *transmissions*, are produced using different technologies (e.g., Real Networks, Windows Media etc.) and transport parameters (e.g., bit-rates and formats). These transmissions are selected to match the expected capabilities of the audience.

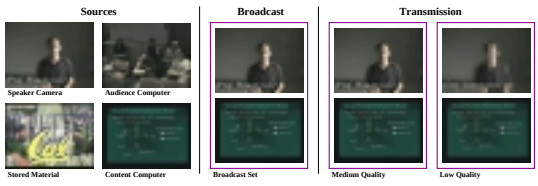

Figure 3: The webcast production model.

3. INTERNET WEBCASTING SYSTEM ARCHITECTURE

The Internet webcasting system architecture is based on the premise that producing a webcast is too complex and computationally intensive for a single application or processor. A collection of distributed processes is required. The webcasting service model divides the processes into clients and services. Services provide functionality for the webcast. For example, one service can be responsible for controlling a camera while another service is responsible for computing video effects. Client applications are processes that use services.

Figure 4 shows the distribution of services used to produce the Berkeley MIG Seminar webcast. Notice the similarity between the software architecture shown in this figure and the physical infrastructure shown previously in figure 2. *Dc* is a client application that uses services to produce the webcast. The specific services shown in the figure are described below. Services send streams into the studio session. *Dc* receives the streams in the studio session, selects a subset for webcast, and transmits those streams into the broadcast session. `Rtc` services are transcoders that forward and optionally convert streams in the broadcast session to the transmission sessions.

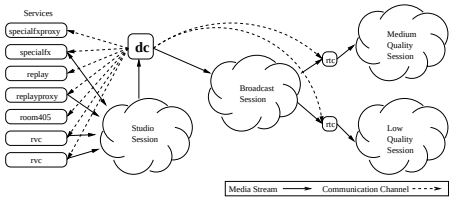

Figure 4: Webcast Architecture.

The distributed service architecture has many benefits over a monolithic application. First, independent services can be developed and deployed rapidly since modifications are not required to existing services. Second, clients adapt to the existence of specific services. They discover services dynamically so they can accommodate a constantly changing infrastructure. Third, the services used by clients can change from one webcast to the next. Fourth, a client depends on the collection of services but not on one service alone. A fault with a single service will not cripple the client. And lastly, processing load can be distributed to a set of commodity machines instead of a single server.

The webcasting service model is an extensible architecture that supports integration and control of services. New services are integrated into the infrastructure through a discovery protocol. Clients follow this protocol to locate services. The model also specifies protocols to incorporate user interfaces that can be used to control the service.

3.1 Service Model

This section describes the protocols that embody the service model. First, a general description of the service architecture is presented. Second, the service discovery

protocol is explained. And lastly, we explain how a service provides a control interface to a client application.

The service model is composed of three components: service abstractions, a service discovery service, and a client API. Services are processes that provide useful functionality for clients. Clients are processes that use services. The Service Discovery Service (SDS) is a directory service that allows a client to locate desired services. For example, *dc* discovers available services when it is executed and uses services selected by the webcast. A simple webcast may only need a single camera service while the Berkeley MIG Seminar webcast requires two camera services, an archive playback service for the opening segment, and a special-effects service. The service model allows *dc* to configure the services required for the particular broadcast.

SDS is a service that lists available services in the infrastructure through a soft-state protocol [17]. A push-pull protocol is used to gather and retrieve information. Services push information to the SDS service and clients pull information from the SDS service. Idle services periodically announce their presence to a well-known multicast address. This information is cached by the SDS service. A timer is associated with the information when it is received. The timer is decremented every half-second. Periodic updates by the services reset the timer. If the timer goes to zero, the service is assumed to be unavailable (i.e.- killed or busy).

Clients query the SDS service cache to locate a desired service. The query can request a specific service or a collection of services (e.g., "all services in 405 Soda"). The SDS process responds with data that satisfies the query. Clients periodically query the SDS service to discover new services.

The service information returned to the client includes the contact and attribute information required to access and use the service. Service attributes provide information about the service (e.g., location, functionality, etc.) and are used to distinguish service types. Contact information includes an IP address and port number pair that can be used by the client to initiate communication with the service.

A client application uses the IP address and port number pair to create a TCP connection with a service. An ad-hoc, non-blocking Remote Method Invocation (RMI) mechanism is implemented that allows the client to invoke methods on service objects.

After initializing and establishing communication with a service, the client application displays a user interface to control the service. The control interface must be provided to the client by the service. Otherwise, the introduction of a new service will require either: 1) that the client source code be modified or 2) the service control interface be limited to a few predefined interfaces. Our experience with webcasting is that the infrastructure varies from room-to-room and is constantly changing. Consequently, the service interface must be incorporated into the definition of the service.

Dc uses the `GetUIControl` method to obtain the interface code segment from the service. This code segment when evaluated will display the control user interface and handle input. *Dc* is implemented in Tcl/Tk so the code segment is a script that creates the interface and input event mappings. The code segment has two arguments: a window and a socket. The service interface widgets are displayed in the window, and the socket is the TCP connection to the service process. *Dc* supplies the arguments when it evaluates the code segment. The socket is used to handle user interface events. For example, a `StartReplay` method is invoked the on a *replay* service object when the `Start` button is pressed in the `replay` dialog box shown in figure 7.

3.2 Webcast Services

This section describes the services listed in table 1 that we developed to produce the Berkeley MIG Seminar webcast. The Remote Video Capture service (*rvc*) controls a video capture device. This service is essentially a server version of the *vic* Mbone tool [12]. *Rvc* encodes analog video signals into RTP packets and provides format and bit-rate interface controls. For the MIG Seminar webcast, two *rvc* services are used to produce two streams. The studio classroom has two capture machines each connected to an AMX control system[3] and an audio/video matrix switcher. Audio/video devices are connected to the AMX system and the matrix switcher. *Rvc* uses an RPC interface to issue commands through the AMX system to the matrix switch and media devices. For example, the *rvc* service can send commands to the AMX system to switch the video feed from the VCR to the speaker camera and commands to move the speaker camera to the right.

The *room405* service provides an interface to non-media services in the classroom that can be operated through the AMX system. The current services include: room lights, raising/lowering projection screens, and turning on/off an "On-Air" indicator light outside the classroom.

The *replay* service manager and *replayproxy* services allow archived material to be played into a webcast. A short opening video segment (approximately 20 seconds) is played at the beginning of the webcast. To play this opening, the director contacts the *replay* service manager and specifies the video to be played. The *replay* service instantiates a *replayproxy* service which is responsible for controlling this particular video. We currently use the MARS multimedia archive system to archive and play stored material on-demand [18]. The *replayproxy* service starts a MARS playback process, connects to *dc*, and acts as a proxy between *dc* and the MARS server. The proxy implements the VCR controls presented in the *replayproxy* control interface.

The *specialfx* service manager and *specialfxproxy* services are similar to the *replay* service manager and *replayproxy* services except they control video effects processes. The *specialfx* service manager starts the *specialfxproxy* service that starts video-effects processes [10]. The proxy service acts as an intermediary between the client application and the processes that implement the special effect.

[3] http://www.panja.com/integrator/amx/

Service	Lines	Functionality
rvc	1768	Capture Machine. Digitizes analog audio and/or video, compresses it, and frames the digital media into RTP packets. Provides controls for both the digitization process and the device(s) connected to the capture machine (e.g., camera, mixer, VCR).
room405	229	Room 405. Controls environment parameters in the room (e.g., light controls, screen raise/drop).
replay	320	Archival Playback Manager. Archive catalog interface and playback service management.
replayproxy	612	Archival Playback Service. Streams archived media into a session. It provides controls to play, pause, and seek in the video.
specialfx	365	Special Effects Manager. Effects system manger and interface for specifying effects services.
specialfxproxy	1402	Special Effects Service. Controls a particular effects service.
rtc	411	Trancoding Service. Transcodes media streams at different bit-rates and formats from one session to another.
null_service	202	Overhead needed by all services.

Table 1: Table of services with their line numbers and functionality.

The *rtc* service is a transcoding service that is still under development. It will take streams from one multicast session, transcode them to different bit-rates and formats, and send them into another multicast session. *Rtc* is a server version of the rtpgw service [2].

4. *DC* APPLICATION

This section describes the user interface of *dc*. The interface organization is shown in figure 5. The main window contains four panels: sources, preview, broadcast, and transmission. The sources panel displays sources in the studio session. The preview panel allows the director to preview and prepare sources before they are webcast. Streams in the webcast are displayed in the broadcast panel. The transmission panel, which is in development, will control webcast transmissions. An example is displayed in figure 6 where a screen dump of *dc* is shown.

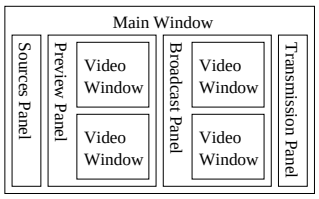

Figure 5: *Dc* user interface organization.

This section begins with a discussion of the sources panel. Second, the video window abstraction used in the preview and broadcast panels is described followed by a description of these two panels. Lastly, we discuss the integration of video-effects into *dc*.

4.1 Sources Panel

The sources panel serves as a starting point for the webcast. It contains mechanisms to start and display sources transmitting into the studio session.[4]

[4]In the television community, a source is defined as cameras and other devices that produce video signals. In webcasts, however, the medium of transport is packets rather than analog signals. Sources are therefore better defined to be packet streams. Under this definition, the capture machine in figure 2 is the source, not the cameras or other devices connected to the matrix switch.

Figure 6: A screen dump of the Director's Console during a production.

A column of small, slowly refreshing thumbnail images represent sources as shown in figure 6. Below these thumbnails are buttons that represent control services. Services perform computations or actions on behalf of an application. These services can be media or control services. A *media service* produces a packet stream (e.g., audio or video). A *control service* provides an interface to control an entity (e.g., room lights, a floor control application, etc.).

The `Service` pull-down menu at the bottom of the sources panel allows the director to add a media or control service interface to the sources panel. The video sources displayed in the sources panel are called the *Media Service Set*. The control interfaces are called the *Control Service Set*. The *Control Service Set* shown in figure 6 includes three services (i.e.- `Room405`, `SpecialFx`,

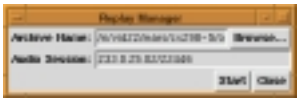

Figure 7: An example control service dialog. This dialog allows the director to start an archive replay source. The two entry boxes specify the archive material and audio session.

and `Replay`) that display dialog boxes to control entities or instantiates a media service. The `Room405` button displays various room controls (e.g., light controls, projection screens). The `SpecialFx` button displays a dialog that instantiates a video-effect services and adds it to the *Media Service Set*.[5] The `Replay` button, shown in figure 7, allows the director to add a stored material playback stream to the *Media Service Set*.

Services are presented to the director through the `Service` pull-down menu, which displays a hierarchical menu based on the service location. The hierarchy has the following structures:

`organization/building/room/service`

and

`Locationless/service`

Figure 8 shows the hierarchical structure for two services.

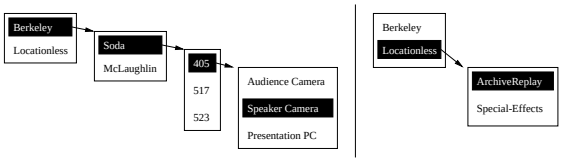

Figure 8: Examples of service menu hierarchical structure. The hierarchy on the left shows a location dependent service and the hierarchy on the right shows a location independent service.

Selecting a menu item initiates the service. A media service transmits a stream to the studio session when initiated. If it is a video stream, a thumbnail computed from the stream is added to the sources panel. A control service adds a button to the *Control Service Set* displayed below the column of thumbnails.

The `Service` pull-down menu changes to reflect the current environment. If, for example, a remote camera is installed during a webcast, the service process associated with the camera will announce its existence. The *dc* will detect this new service through the service discovery protocol and add it to the `Service` pull-down menu.

4.2 Video Window

Thumbnails provide an excellent summary of available video sources, but a director typically wants to preview and prepare the stream before switching to it (e.g., position a replay). Consequently, *dc* has preview windows for streams being prepared. The previews are displayed as a video window in the preview panel immediately to the right of the sources panel. Video windows are also used to display streams in the broadcast panel.

A video window displays a CIF-sized image at higher frame rate than a thumbnail.[6] The interface controls below the video window allow the director to control the stream and source being displayed. The specific control interface depends on the source. For example, the control interfaces for a computer-controlled camera includes focus and position controls (e.g., pan/tilt or left/right position). A playback source has VCR-controls (e.g., play, pause, position, etc.). Other video sources have different control interfaces depending on the source.

The control interface will change when the service state changes. Consider the capture machine service connected to the matrix switch in figure 2. It can produce a stream from any one of many devices, each of which may have its own controls. Consequently, the capture machine control interface needs controls for the video switch and controls for the currently selected video device. When the director switches the capture machine from the VCR to a camera, camera interface controls replace the VCR controls below the video window.

4.3 Preview and Broadcast Panels

To prepare a source for broadcast, the source is first displayed as a video window inside the preview panel. A source can be previewed by selecting a thumbnail with the mouse, dragging it to the preview panel, and dropping it. Dropping the thumbnail on an existing video window displays the new source in place of the previously displayed source. More than one video window can be displayed within the preview panel. Dropping the thumbnail in the preview panel but not over a video window instructs *dc* to create a new video window and display the source in it. A video window can be removed from the preview panel by selecting it with the mouse, dragging it outside the preview panel, and dropping it.

The broadcast panel has the same interface as the preview panel. A video window displayed in the broadcast panel, however, is a stream in the webcast. Data packets from streams in the broadcast panel are relayed through *dc* into the broadcast session where audiences view the webcast. Multiple streams are webcast by having several video windows within the broadcast panel. The director switches the source in the webcast stream by dragging and dropping a thumbnail or video window in the preview panel to the broadcast panel. The broadcast panel behaves the same way as the preview panel - dropping on a video window switches the stream and dropping elsewhere creates a new video window (i.e., a new stream in the webcast).

4.4 Video-Effects Sources

[5] A service is created for each special-effect that can be applied during a webcast. Multiple instances of an effect can be created and used simultaneously.

[6] Thumbnails are updated once per second and video windows are updated as specified in the stream being displayed.

Television video-effects are implemented by a video production switcher (VPS). A VPS can switch from n inputs to one output that is typically the broadcast output. A VPS can apply effects on the input(s) as the data is routed to the output. Commercial systems use custom-designed hardware to meet the processing requirements of broadcast quality video. Some VPS systems can layer effects on top of other effects (e.g., fading from one stream to another with titling). However, the number of layers is bounded, and most VPS systems do not allow new effects to be added.

The Parallel, Software-only Video-effects Processing System (PSVP), which runs on a Network of Workstations, was developed by our research group to provide a low-cost, extensible video-effects system [10]. PSVP produces an effect by decomposing it into a collection of processes that read RTP packets from a multicast session, compute an effect, and write RTP packets back into the multicast session.

Dc invokes and controls an effect through the service infrastructure. Effects services (see figure 2) input streams from the studio session, render an effect, and output a new "effects stream." Effects streams are ordinary sources as far as *dc* is concerned with controls for the effect. Thus, effects can be cascaded by piping the output of an effect to the input of another effect service. A titling effect is shown in the Berkeley MIG Seminar webcast shown in figure 1. Adding a new effect to the webcast requires only that a new service be installed in the infrastructure.

5. DESIGN AND IMPLEMENTATION

Dc and the service processes are implemented as Mash applications. Mash is a multimedia toolkit for developing distributed collaboration applications [13]. It provides the basic objects needed to create multicast applications (e.g., media stream receivers/transmitters, media codecs, etc). Mash is a split architecture software system. Routines that are performance sensitive (e.g., media coding and decoding, network transmitting and receiving, etc.) are implemented in C++. User-interface abstractions and performance insensitive code is implemented in the Tcl/Tk scripting language. Mash also uses the Object Tcl (Otcl) extension for object-oriented programming.

Dc and the service programs make extensive use of the following packages written in Mash:

- *Active Service Framework.*
  The AS framework provides a programmable substrate on which to build arbitrary network services [1]. These services reside in one or more pools of active service nodes. Client applications instantiate service agents on one or more nodes. The framework ensures fault-tolerance and load-balancing for clients. The *replayproxy* and *specialfxproxy* services use the AS framework to instantiate MARS and PSVP system services, respectively.

- *Scalable Naming and Announcement Protocol.*
  SNAP is a framework for selective reliable multicast. It provides a hierarchical naming system and a scalable session announcement protocol [16]. The SDS service uses SNAP to deliver reliable messages between clients, services, and SDS servers for the service architecture. In addition, SNAP provides containers for messages. Container names are organized into a hierarchical name space. Clients connect to a SNAP multicast and session register callback procedures for containers when a message is transmitted to that container. This facility supports a general-purpose filtering mechanism for messages sent to a multicast session. The webcasting service architecture uses `UPDATE`, `QUERY_REQUEST`, and `QUERY_RESPONSE` SNAP containers for service announcements, client queries, and SDS service query results, respectively.

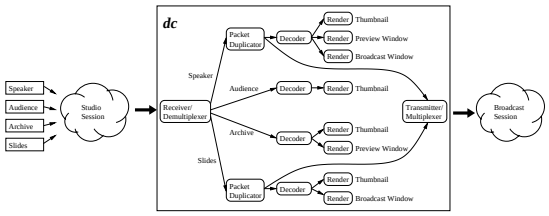

Figure 9: *Dc* Architecture.

The internal architecture of *dc* is shown in figure 9. A network agent receives packets from the video studio session and demultiplexes them to a handler for each stream. A stream may be displayed in one or more windows and, if it is in the broadcast set, the packets are sent to the multicast session for the broadcast. A stream is always displayed in a thumbnail so the packets must be decoded and rendered to a thumbnail window. If the stream is also being displayed through one or more video windows, the decoded packets are passed to additional renderers for each window. Packets for streams selected for broadcast are duplicated before they are passed to the decoder. The duplicate packets are sent to the broadcast session.

The *dc* application is approximately 2,800 lines of code, which includes 430 lines of C++ and 2,400 lines of Tcl/Tk code. C++ is used to duplicate packet buffers for broadcast and relay them to the transmitter. The breakdown of the Tcl/Tk code is shown in table 2. The code length does not include inherited objects and consequently may appear skewed. For example, the `agent-broadcast` object inherits transmission functionality from the MASH `video-agent` object. So even though *dc* only shows 48 lines of code, `video-agent` includes over 1,000 lines of code.

Services are implemented only with Tcl/Tk code. Table 1 above shows the number of lines for each service. The `null_service` shows the code needed by all services. It contains code to interface with the SDS service and clients but has no functionality. The proxy services act as intermediaries between clients and backend

Name	Lines	Functionality
`agent-broadcast`	48	Network agent transmitting to the broadcast session.
`agent-studio`	119	Network agent responsible for receiving packets from the studio session and demultiplexing the packets into its constituent streams.
`application-dc`	224	Application object responsible for creating and organizing other objects in the *dc* .
`session-dc`	129	Objects used to discover services.
`ui-dc`	118	The main window object of *dc*. Creates the panel objects that reside within it.
`ui-dcbroadcast`	212	Object responsible for broadcast panel.
`ui-dcpreview`	177	Object responsible for preview panel.
`ui-dcthumbnail`	475	Object responsible for thumbnail panel. It is responsible for new services.
`ui-dcthunbnailservice`	284	Control service objects.
`ui-dcthunbnailvideo`	89	Media service objects.
`ui-dcvideo`	248	Video Window Object.
`dc-link`	324	RMI mechanism used to send commands between *dc* and services.

Table 2: Table listing *dc* code. Each object name is followed with code length and a summary of its function.

servers which explains why `replayproxy` can be written in only 612 lines. The majority of the `replayproxy` service is in the MARS backend process. This example shows the power of Tcl/Tk as a "glue language" for building interfaces between disparate applications.

6. DISCUSSION AND FUTURE WORK

A quick prototype for *dc* was implemented by Wu in the Fall, 1998. It provided a remote control interface to the AMX control system so the Berkeley MIG Seminar broadcast engineer could control which video source was routed to the capture machines. The need for a new *dc* that supported multiple streams, interfaces to more services, and dynamic extensibility was immediately obvious. The design and implementation of the current version of *dc* was completed during the Spring and Summer of 1999. This version was used to produce the seminar webcasts beginning in the Fall, 1999 semester. Although the system has performed well beyond initial expectations, it can still be improved. This section discusses several changes and improvements.

A webcast becomes more and more difficult to manage as the number of services increases. The special-effects service alone can double the number of input streams and interface controls. Our goal is to have one person produce several webcasts simultaneously. Consequently, the need for automation is apparent. Automation eliminates the need for the director to perform mundane manual tasks. One particularly time consuming chore is camera tracking. As the speaker moves about the stage, the camera must be moved to ensure he or she is correctly framed. Automatic camera tracking is a well-known technique for solving this problem that should be incorporated into the webcasting system [7].

A second example is stream switching. Take the case of a webcast composed of speaker and content streams. If an audience member asks a question, the content stream should be switched to the audience camera and the camera should be moved to show the person asking the question. The fact a question has been asked is known because the audio mixer detects that the input from an audience microphone is above a threshold. Our studio classroom has three audience microphones mounted in the ceiling on the left, center, and right sides of the room. The mixer adds the output of the speaker microphone to the room audio and routes it to the webcast audio stream. Consequently, the mixer can identify the side of the room from which the question comes. We need a mechanism for the mixer to signal the webcast control system that an audience question from a specific area of the room has occurred. An intelligent control system that implements a policy specified by the director can automatically execute the commands to switch to the audience camera and position the camera to the appropriate area in the room. In addition, the bandwidth allocation between the speaker and content stream can be changed to reflect this new configuration. Low-bandwidth transmissions allocate more bandwidth to the speaker because slides are often static relative to the speaker. However, bandwidth should be diverted from the speaker to the content stream when switching from slides to the audience. Lastly, automation can be used to log events, such as stream switching, to assist video query, summarization, and automated authoring of multimedia titles derived from a lecture [15].

Automation relies on two mechanisms: 1) the ability to monitor events produced by other processes in the webcast and 2) the ability to invoke commands in lieu of the director. The bandwidth-allocation agent, for example, monitors switching events and responds with commands to the transcoder to either increase or reduce bit-rates. The current *dc* architecture cannot support such agents because it uses unicast communication channels to services. Thus, communication, both events and commands, between clients and services cannot be monitored or forwarded to other participants. However, using multicast for the communication channel will remove this limitation. An appropriate logical decomposition of the events can be created using SNAP containers so automation agents can monitor only the events in which they are interested. The bandwidth-adjusting

agent can wait for an event like "Switch Content Stream to Audience Camera" and respond with commands to reduce speaker bandwidth and increase content bandwidth.

A second area in which *dc* can be improved is security. We ignored many security issues during the development of *dc* because we used rapid prototyping to the system. In retrospect, we note many security weaknesses that will need to be addressed. First, the SDS service allows any process to present itself as a service. It also gives any process information about all existing services. Rather than duplicating effort, we plan to replace the prototype SDS service with the Ninja SDS System that is better designed for security [6]. A second security issue arises because Tcl/Tk code is transferred from services and evaluated in the client without checks. Thus, a malicious service can kill a client application by simply sending an exit command. Moreover, the Tcl/Tk code segment is difficult to write correctly. So, even a trusted service might cause harm through carelessness. One approach to solve these problems is to implement a certificate system that ensures the correctness and trust of the code. Another approach is to create a description language for user-interfaces similar to the document language proposed by Hodes and Katz [9]. Rather than sending code, services transfer a description of the user-interface. By restricting the description language, clients can limit the access to services and thereby protect themselves. A third approach is to sandbox the code as is done in Janus [8]. The code runs in a confined area with limited access to resources (e.g., socket communications can be limited to the parent service or file writes can be prohibited).

A third area in which *dc* can be improved is in audio control. Audio has been largely neglected by the current *dc* prototype. However, many of the design principles used for video also apply to audio. Audio devices can be viewed as a service that can be switched in conjunction with or independently from video. Audio effects can be applied just as in video. However, the biggest problem we must solve is to provide seamless interaction with remote participants. Solving this problem will require changes in the studio classroom to provide visual feedback or information about remote participants (i.e., to provide a sense of presence [4]) and an audio control system that can handle echo cancellation with long delays resulting from Internet communication.

A fourth area in which *dc* can be improved is controlling commercial webcast transmissions. We began simulcasting the Berkeley MIG Seminar using Real Networks because the unicast transport used in commercial systems is easier to deploy than the multicast transport used in the Mbone. Integrating these technologies into *dc* will enable a wider webcast audience. In addition, viewers will benefit from the flexibility and versatility of *dc*. For example, the Synchronized Multimedia Interaction Language (SMIL) developed by the World Wide Web Consortium can be used to view webcasts with multiple streams like those produced by *dc* [22].

A fifth area in which webcasts produced by *dc* can be improved is interactivity. Many webcasts follow the television model of broadcasting in which there is no audience interaction. The Internet, however, is capable of audience participation. Nowhere is this capability more important than in distance learning where the student's ability to ask questions during a lecture is critical. Many mechanisms for student participation exist (e.g. chat sessions, video conferencing, and interactive response systems), but more research is needed to determine the most appropriate models for integration with live webcasts. Moreover, moderation techniques will need to be developed to handle non-trivial numbers of participants. For example, if 200 people want to ask a question simultaneously, who gets control and how is the interaction moderated?

Lastly, our experience suggests that future studio classrooms should use equipment directly connected to the Internet. Rather than connecting cameras and microphones to matrix switches and mixers and using a control system like the AMX, a better solution will connect the devices directly to the network. Switching and control are easier to implement. And, production control will be better if all devices are available at any time rather than just a subset of devices through a matrix switch.

7. RELATED WORK

This section describes related work on broadcast production and control.

Several projects have developed software interfaces to a video production system. NBC's *GEnesis* project monitors and controls program streams with a software system built with Tcl/Tk [3]. *GEnesis* provides a graphical user interface to control broadcast schedules that vary over time and region. For example, a live sports event has commercials that vary depending on the location. The system employs a database to store and retrieve schedules, and device controls are accessed with a string protocol over sockets. Operations are implemented by sending commands to conventional broadcast equipment. Both *dc* and *GEnesis* control remote devices, but they are solving different problems. *Dc* concentrates on the production of video streams rather than the scheduling of existing streams.

The Berkeley Multimedia Research Center developed several prototypes before *dc*. The *Software-Only Video Production Switcher* project introduced video-effects processes to our production environment [21]. It used remote processes that received and transmitted streams from a studio session. It employed a message protocol to control these remote processes. However, the user interface was hard-coded into the broadcast application and there was no notion of modular services. Moreover, only a single stream was produced from the system. This system was an early prototype for the PSVP effects system.

8. CONCLUSION

Fundamental differences between the underlying infrastructure of webcast and traditional television prohibits using a television production model for webcasts

without modification. We propose a webcast production model that addresses these differences. The model is composed of three stages: *sources*, *broadcast*, and *transmission*. We have developed the Director's Console, a webcast production tool, to demonstrate the effectiveness of this model. *Dc* is of a distributed client/server system organized under a webcast service architecture. The service architecture provides fault tolerance and dynamic adaptation to a changing physical infrastructure.

9. ACKNOWLEDGMENTS

We would like to thank Irene Hsu, Peter Pletcher, Oliver Crow, Paul Huang, Tim Fitz, and the rest of the BMRC Researchers for their invaluable help during development of the project. Thanks also go to Richard Fateman, Chris A. Long, and other readers for their input on this paper.

10. REFERENCES

[1] E. Amir, S. McCanne, and R. Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. In *Proc. of ACM SIGCOMM '98*, Vancouver, Canada, August 1998.

[2] E. Amir, S. McCanne, and H. Zhang. An Application-level Video Gateway. In *Proc. of ACM Multimedia 95*, San Francisco, CA, November 1995.

[3] S. Angelovich, K. Kenny, and B. Sarachan. NBC's GEnesis Broadcast Automation System: From Prototype to Production. In *Proc. of the 6th Annual Tcl/TK Conference*, 1998.

[4] D. Boyer and M. Lukacs. The Personal Presence System – A Wide Area Network Resource for the Real Time Composition of Multipoint Multimedia Communications. In *Proc. of the Second ACM International Conference on Multimedia '94*, pages 453–460, San Francisco, CA, October 1994.

[5] D. Culler, et al. Parallel Computing on the Berkeley NOW. In *9th Joint Symposium on Parallel Processing*, 1997.

[6] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. In *MOBICOM '99*, Seattle, WA, August 1999.

[7] D. B. Gennery. Visual Tracking of Known Three-Dimensional Objects. *International Journal of Computer Vision*, 7(3):243–270, April 1992.

[8] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A Secure Environment for Untrusted Helper Applications: Confining the Wiley Hacker. In *1996 USENIX Security Symposium*, 1996.

[9] T. Hodes and R. Katz. A Document-based Framework for Internet Application Control. In *2nd USENIX Symposium on Internet Technologies and Systems*, Boulder, CO, October 1999.

[10] K. Mayer-Patel and L. A. Rowe. Exploiting Spatial Parallelism for Software-only Video Effects Processing. In *Proc. of The Sixth Annual ACM International Multimedia Conference*, September 1998.

[11] S. McCanne. Scalable multimedia communication with internet multicast, light-weight sessions, and the mbone. *IEEE Internet Computing*, 3(2):33–45, 1999.

[12] S. McCanne and V. Jacobson. *vic*: A Flexible Framework for Packet Video. In *Proc. of ACM Multimedia 95*, San Francisco, CA, November 1995.

[13] S. McCanne, et al. Toward a Common Infrastructure for Multimedia Networking Middleware. In *Proc. of NOSSDAV 97*, St. Louis, Missouri, August 1997.

[14] G. Millerson. *The Technology of Television Prodcution*. Butterworth-Heinemann, 12th edition, March 1991.

[15] S. Mukhopadhyay and B. C. Smith. Passive Capture and Structuring of Lectures. In *Proc. of ACM Multimedia 99*, pages 477–487, Orlando, FL, November 1999.

[16] S. Raman and S. McCanne. Scalable Data Naming for Application Level Framing in Reliable Multicast. In *Proc. of ACM Multimedia '98*, Bristol, UK, September 1998.

[17] S. Raman and S. McCanne. A Model, Analysis, and Protocol Framework for Soft State-based Communication. In *Proc. of ACM SIGCOMM '99*, Cambridge, MA, September 1999.

[18] A. Schuett, S. Raman, Y. Chawathe, S. McCanne, and R. Katz. A Soft-State Protocol for Accessing Multimedia Archives. In *Proc. of NOSSDAV 98*, Cambridge, UK, July 1998.

[19] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RFC 1889, RTP: A Transport Protocol for Real-Time Applications*, January 1996.

[20] The Standard. Streaming media numbers start to flow. http://www.thestandard.com/article/display/0,1151,8419,00.html.

[21] T. Wong, K. Mayer-Patel, D. Simpson, and L. A. Rowe. A Software-Only Video Production Switcher for the Internet MBone. In *Proc. SPIE-Multimedia Computing and Networking 1998*, San Jose, CA, January 1998.

[22] World Wide Web Consortium. *Synchronized Multimedia Integration Language (SMIL)*, November 1999. http://www.w3.org/AudioVideo/.

[23] D. Wu, A. Swan, and L. A. Rowe. An Internet Mbone Broadcast Management System. In *Proc. of Multimedia Computing and Networking, SPIE 99*, San Jose, CA, January 1999.