Zoned Analog Personal Teleconferencing (ZAPT)

Joseph D. Touch[1]
USC / Information Sciences Institute
touch@isi.edu

ABSTRACT: ZAPT is a desktop multiparty audio/video teleconferencing system at ARPA, based on Bellcore's Touring Machine. ZAPT supports one multiparty call (up to 4 parties), and any number of point-to-point calls, with analog multimedia. ZAPT supports automated remote operation and conference "join" requests. ZAPT provides interoffice conferencing at ARPA, and manual cross-connect to Internet IETF-style conferencing tools.

1.0 Introduction

This is a report on the Zoned Analog Personal Teleconferencing (ZAPT) project, to port Bellcore's Touring Machine (the TM) analog video teleconferencing system to the NeXT Cube computers at ARPA. ZAPT was a one-year effort, begun in mid-June 1992. ZAPT has been operational since February 1993, with enhancements concluded in April 1993. The primary goal was to provide desktop teleconferencing at ARPA in minimal time, using ARPA's existing environment. ZAPT uses Bellcore's Touring Machine with extensions, including passive bridging, fault management, a NeXT-style user control interface, and automated clients. ZAPT provides interoffice desktop video teleconferencing, and manual cross-coupling to existing Internet teleconferencing. The crossbar control interface and bridging extensions were shared with the research community. ZAPT also influenced the multi-way communication models of our Multi-Media Conferencing (MMC) and Scalable Personal Teleconferencing (SPT) projects. Our conclusions include comments about off-site interoperation, fail-safety, and security. The NeXT was also evaluated as a potential digital A/V teleconference platform. We recommend eventual replacement of ZAPT with a system designed for off-site call capability.

2.0 Goals

The goal of ZAPT was to permit ARPA to reap the benefits of teleconferencing projects it has supported, for internal use, as well as to showcase the technology ARPA supports, using "off-the-shelf" components and software. ARPA has funded various wide-area teleconferencing projects, including those that use the TWBNet/DSINet [Ca90] and DARTnet networking testbeds, as well as supporting components of other teleconferencing projects. Each of these projects has its own specific goals, primarily related to technology development or servicing external users. In early 1991, when ZAPT[2] was developed, the existing ARPA projects were research only, and not appropriate to support impromptu teleconferencing, in a "no white-coats"[3] style. While commer-

1. This research was sponsored by the Advanced Research Projects Agency through Ft. Huachuca Contract No. DABT63-91-C-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Army, the Defense Advanced Research Projects Agency, or the U.S. Government.

2. ZAPT was originally called "ACT- Automated Cluster Teleconferencing". The name was changed to distinguish it from the ACTS project, which involves satellites.

3. "no white-coats" implies user-managed operation. No lab personnel or technicians should be required.

cial providers of teleconferencing existed, the equipment and setup costs were prohibitive, and they were designed for conference room use, rather than desktop teleconferencing.

ZAPT's primary goal was high-quality, low cost, multiparty desktop teleconferencing at ARPA. ZAPT uses existing software, to minimize delivery time, and also uses existing hardware at ARPA, specifically the NeXT systems and analog office cabling. Security was also a goal, and interoffice analog wiring was deemed sufficiently secure.

3.0 Description

Here we describe the ZAPT system, which is composed of a modified version of Bellcore's Touring Machine, with bridging and switching extensions, as well as modified automated user clients. The discussion includes a system overview, description of the user interface, and summary of internal modifications that affect the system behavior.

3.1 System Overview

The Touring Machine is an analog multimedia teleconferencing system based on centralized control, developed at Bellcore [Ar92] [Ar93]. At the time ZAPT began, the TM was the existing system that most closely matched ZAPT objectives. Bellcore currently uses the TM for teleconferencing among offices within one site and between two sites 50 miles apart. Bellcore supported efforts at MIT and Univ. of Wisconsin to port the TM to other computers, and was willing to release its software for ARPA use.

TM contains two user interfaces: MTS (low-level) and CRUISER (high-level), implemented in X11, on Sun-3 desktop computers, with independent analog audio and video cabling. MTS was chosen because it was more portable, as we will discuss later. The TM manages the analog connections using switching and bridging equipment residual from prior testbeds at Bellcore. The ZAPT system is shown in Figure 1, and its architecture is dominated by the TM components. The TM provides an analog local distribution system, which ZAPT augments with back-end connections to other conferencing systems via the Internet.

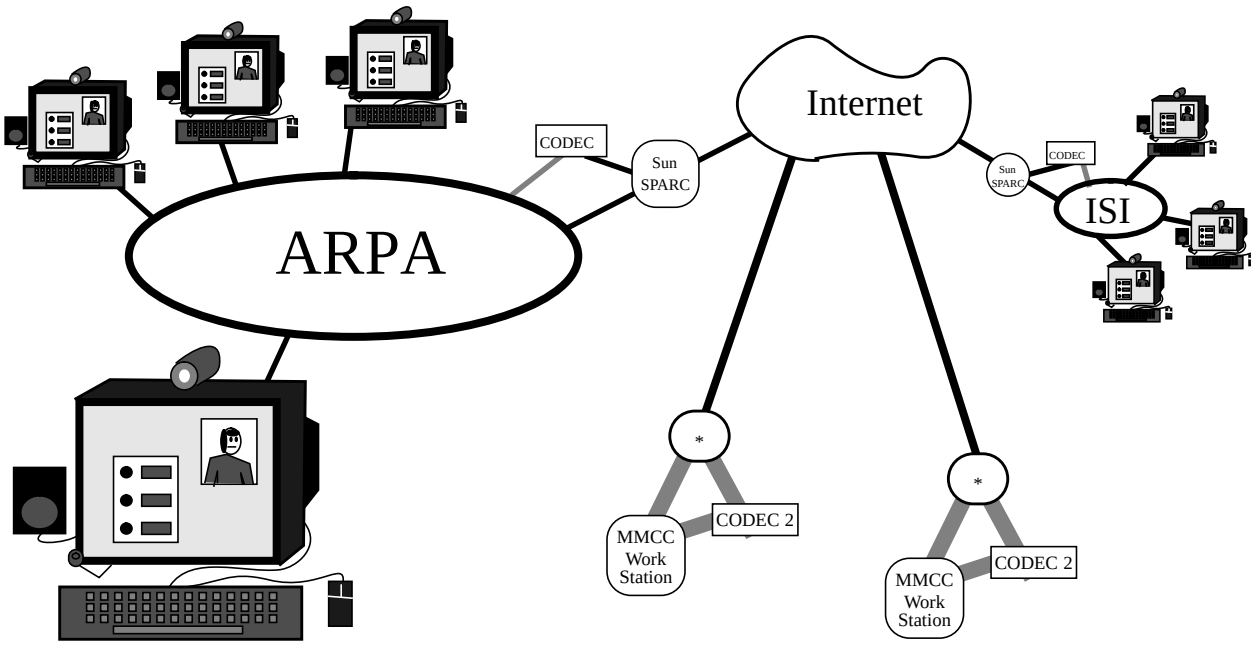

FIGURE 1. ZAPT system overview

In the TM, existing IP transmissions (via Ethernet) are used for control, and an analog crossbar provides audio and video switching. At Bellcore, Sun-3's are used to control separate user-end

analog equipment (camera, monitor, microphone, speaker). At ARPA, the analog video monitor is provided by the NeXT via the NeXTDimension board, using the NeXTtv application to display real-time NTSC video on the color monitor (Figure 2). At both Bellcore and ARPA, analog signals are carried by a distinct cabling network, and digital and analog interact only at the crossbar switch. The bridging at ARPA is completely passive (analog signal independent mixing only), whereas Bellcore's is active (includes signal-dependant mixing as well as switching).

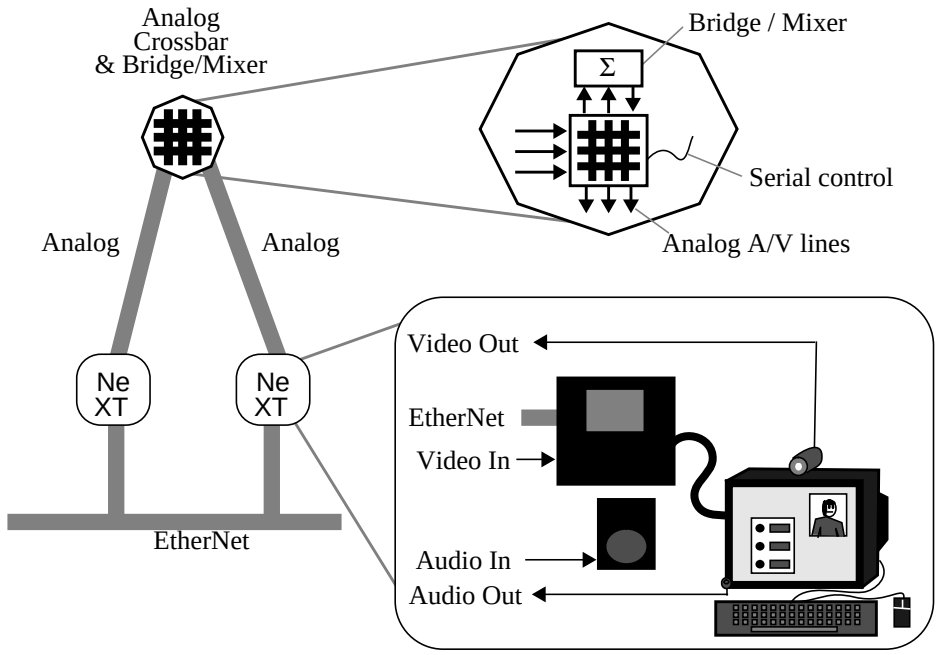

FIGURE 2. ZAPT NeXT components

3.2 User interface

The ZAPT user interface has four components - the ZAPT Manager, MTS, SMGR, and NeXTtv. The ZAPT Manager is a NeXT-style application, which starts the visible and underlying user client components, replacing Bellcore's command-line startup script. The manager integrates component initiation with failure recovery, and controls the user component processes via Unix signals - one which restarts the user components *(SIGINT)*, and the other which shuts the user components down *(SIGQUIT)*. Figure 3 shows the NeXT icon for this user startup interface. Figure 3 also shows that interface opened, with its menu, and two control buttons (Restart and Quit). Figure 4 shows the "info" submenu, and the information panel menu item, and panel itself.

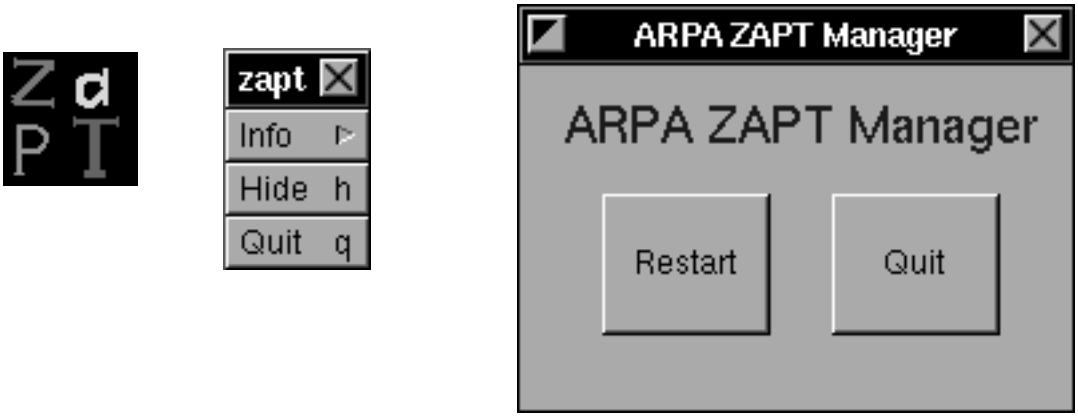

FIGURE 3. ZAPT Manager icon, main menu, and main panel (actual size)

FIGURE 4. ZAPT information panel (actual size)

Analog video is displayed on the NeXT via NeXTtv, a demo application provided with NeXTDimension boards. A sample video window, showing quadrant-split multiparty teleconferencing, is shown in Figure 5. NeXTtv comes up “off” - users must manually “turn on” the tv, using the “power” button in the lower left corner.

FIGURE 5. NeXTtv view of a multiparty teleconference (reduced by 50%)

MTS and SMGR are TM X-windows user interfaces that run directly on the NeXT, under co-Xist (a NeXT application that provides X-windows compatibility). MTS is a Bellcore TM client application for conferencing control designed for computer scientists. Bellcore’s Cruiser client, designed for casual users, was not used because it relies on X-windows extensions not available

under co-Xist. Figure 6 shows a sample of the MTS user interface and SMGR endpoint control tool, both of which are unchanged from Bellcore's Sun-3 version, except for the NeXT-style window manager "iconize" and "close" buttons[1].

The MTS interface provides call initiation and response, as well as multiple call management. A user selects the called parties from a static user list, and initiates a call. Calls can be accepted or refused by the user, or SMGR can auto-accept (discussed below). Calls in progress can be suspended, resumed, or brought to the foreground (in the case of multiple calls in progress). An individual call can have its audio in, audio out, video in, or video out individually activated or deactivated on a per-user basis. A call can also be in "split screen" mode, or point-to-point mode. Calls to more than one other party automatically initiate split-screen mode. Up to 3 other parties can be part of a single call (for a total of 4, including the caller). Only one multi-party call can be in session at a time in this installation of ZAPT, although any number of point-to-point calls can occur simultaneously. In addition, individual users can be added or deleted from a current call, at the discretion of any current member of the call.

In addition to MTS, each station has a control interface, called SMGR, also shown in Figure 6. A station is defined as a set of endpoints, shared among the user client processes. SMGR is used to change the default policy for session requests, from manual, to auto-accept, auto-deny, and pass-through (not used for MTS). It also shows the status of the endpoints of a station.

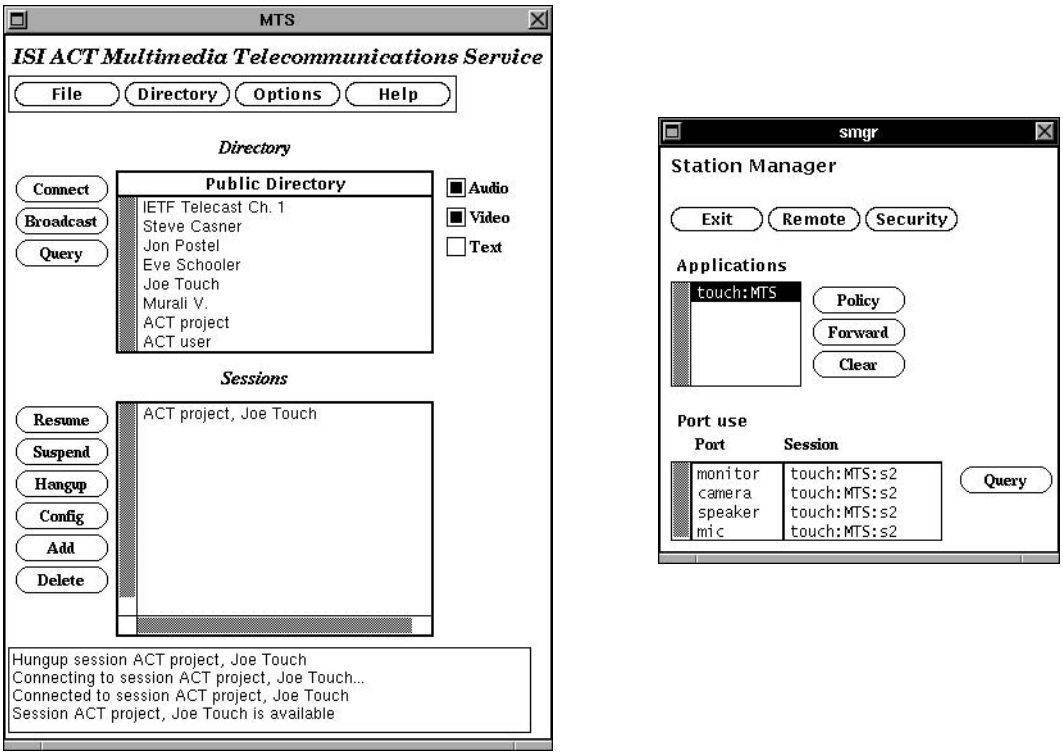

FIGURE 6. MTS and SMGR user interfaces (reduced by 50%)

3.3 Internal modifications

ZAPT includes a number of internal modifications, which support modes of operation beyond that available at Bellcore's Touring Machine installation. These include passive bridging, fault-detecting switching, automated user client operation supporting cross-coupling, and fail-safe user and system component maintenance.

1. A bug in co-Xist causes the "close" button on the NeXT-style windows to quit the user application. co-Xist implements *uwm* semantics, where the close means QUIT, rather than *twm* semantics, where close means CLOSE.

3.3.1 Bridging

Bellcore uses an (expensive) active bridging system as part of their TM installation. Bellcore provided the TM-side of this software, which was modified for use with passive bridging equipment in ZAPT. The video bridge at both ZAPT and Bellcore was passive, and we configured an inexpensive TEAC Tascam studio mixer to provide passive audio bridging in ZAPT as well. Bridging reservation requests are satisfied on a first-come first-served basis.

Active bridging is signal-dependant mixing; in Bellcore's case, the mixed signal output is a "loudest 2" normalized summation of the inputs. In addition, Bellcore's bridging system performs some switching as well. Passive bridging is signal-independent mixing; in ZAPT, each output is an "all but me" sum of the inputs (regardless of signal on those inputs). One advantage of ZAPT's approach is its ability to correctly compose multiple bridges. To compare the two, consider 4 sites in conference, W and X near bridge A and Y and Z near bridge B. Bridge A unites W, X, and the output of B. In Bellcore's case, W hears 0.5 X + 0.25 Y + 0.25 Z, so the remote participants are not as loud. In ZAPT, W hears X + Y + Z, as it would in a conference room, removing only direct feedback. "All but me" bridging is limited to small groups, because background noise is added.

ZAPT is the only bridging TM outside of Bellcore itself. Bellcore's bridge manager software component of the TM performs trunk reordering to undo some of the switching done by their active bridge. ZAPT's bridging software relies on the TM resource allocation mechanism to preserve trunk order, an assumption that may not be valid for multinode switching using Bellcore's version of resource management. Bellcore has reincorporated the ZAPT modifications into their software, and is examining its implications on resource management in the TM.

3.3.2 Switching

The ZAPT switching control software is adapted from code MIT developed for use with their TM installation. ZAPT slows the command stream down with fixed delays to prevent serial line overrun, and verifies all serial port writes with echo-reads, resulting in more reliable operation and fault attribution. Diagnosis of this problem was facilitated by the development of SoftPanel, a *curses*-based (terminal-type independent full-screen ASCII) tool for control of the switch using an ASCII terminal (Appendix C). SoftPanel has been given to the crossbar switch manufacturer, as well as made available on the Internet via anonymous FTP.

3.3.3 Cross-coupling

ZAPT can interconnect with Internet teleconferencing via manual cross-coupling[1]. Side-effect cross-coupling uses an automated TM client and other Internet teleconferencing tools independently. The design is the equivalent of a "null modem" cable (Figure 7). These connections require manual connect of each side of the conference, i.e., a rendezvous at a proxy. Users never receive calls initiated from the proxies, in this case. As far as the TM clients are aware, they have connected to an office called "external". As far as the external client (e.g., MMCC) is aware, the local client has switched the audio and video to a different conference room.

Fully automated cross-coupling would avoid separate control actions on the two conferencing systems. A user on TM calling an external user would result in a call to the proxy, and the proxy would complete the link by initiating an external call to the external user. This requires a connection protocol that is compatible with both TM and the external system semantics. We designed an interoperation protocol for this purpose, which was equivalent to circumnavigating TM entirely, and thus not implemented (Appendix A). A proxy needs to emulate the protocol at the user client level. The client protocol for external Internet teleconferencing tools were documented (e.g., MMCC [Sc92a] [Sc93]), the TM was not. We developed a packet prober to determine the TM protocol, and developed an extension to it for proxy operation (Appendix A). We also developed a protocol for communication between the TM and MMCC clients, equivalent to a 2-party connection protocol (Appendix A).

1. Automated cross-coupling was considered, but deemed beyond the scope of this project.

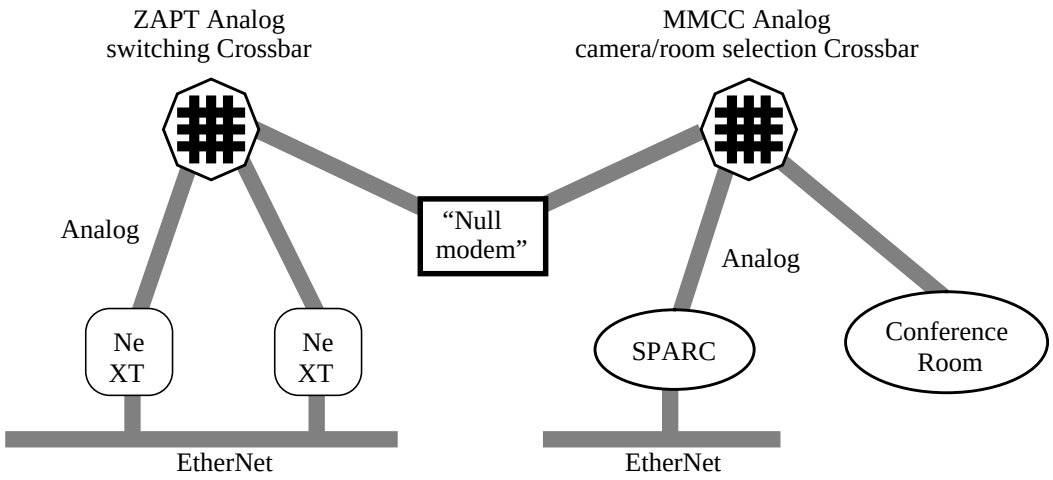

FIGURE 7. Side-effect cross-coupling

ZAPT uses the MTS client, in conjunction with an SMGR client modified to auto-accept, to support call management via side-effect cross-coupling. Side-effect cross coupling is the link formed when two teleconferencing systems call a "null-modem" - the link is a side effect of the call, not due to connection status crossing the link boundary. This supports only the conventional TM call model, where only existing call members can add additional parties to a current call.

ZAPT also supports radio-like broadcasts, in which users join independently via the Radio proxy application. The Radio proxy is an application that replaces MTS. The Radio responds to call requests to add that user to the current broadcast session[1]. The SMGR client was modified to avoid the X-windows interface, to permit automated operation. We used the Bubble trace program (Appendix C) to determine a protocol for the Radio proxy. The Radio proxy was not released in ZAPT, because the protocol was optimistic only (thus not fault tolerant). Further detail of the Radio proxy appears in Appendix A.

We also attempted to augment the Radio proxy to perform both broadcast and point-to-point direct calls. The proxy was to perform dual registration, as Radio and MTS clients; calling the Radio would results in a callback join to a broadcast session, calling the MTS proxy would result in point-to-point auto-accept call operation. The Radio and MTS proxy had to be implemented in a single module, because they share state. When the Radio had a broadcast session active, the MTS proxy should respond "busy" to all requests, and vice-versus. The dual client was not completed, because of the lack of fault tolerance, as noted above.

3.3.4 Failure detection and recovery

ZAPT augments the TM model of system fault-tolerance, because ARPA requires the privacy of fail-stop operation. Failure of the control system should result in failure of the audio and video links; otherwise, transmissions continue without explicit action. Individual components of the TM implement some simple fault tolerance, but due to a lack of soft-resets the set of components is not tolerant of individual component failure. TM's failure mode also keeps a connection up when components fail (fail-stay), rather than keeping them down (fail-stop) as ARPA requires. The TM architecture did not permit a fail-stop design, in general, but a specific version of fail-stop, forcing entire system reboot, was possible. ZAPT also supports self-restart in the event of failure.

We designed SafeNet to manage fault tolerance, and incorporated it into the ZAPT system. SafeNet is a process that monitors the components of the TM, and spawns the failed components in the proper order, either at the system or user level (Figure 8).

1. TM version 2 does not have a session join request.

The TM consists of a core, composed of 7 independent components, and user sites, each composed of 5 components. TM core state information is kept when the core fails, but can't be reset or resynchronized once the TM is running. TM core state information needs to be reset to some known state, but the TM user clients aren't aware of that state. As a result, user clients can fail and restart while the core is running, but core failure requires user clients restart *after* the core restarts. Ideally, the core SafeNet would signal the user SafeNets to restart after it rebuilds the core. In the current system, a user attempting to initiate a call after a core restart is told he doesn't exist, and must infer his own restart action.

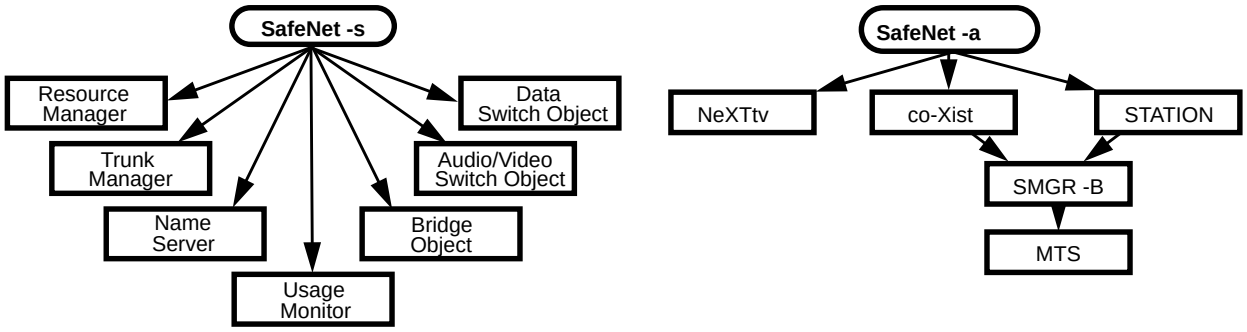

FIGURE 8. TM Core and user clients SafeNet configurations

The core system configuration in Figure 8 indicates that when any system component fails, the entire system is restarted. The user configuration in Figure 8 indicates that the user components exhibit a restart structure, i.e., that MTS can be restarted without restarting other components, but that STATION failure requires restarting SMGR and MTS. The SafeNet software is also integrated into the ZAPT Manager NeXT-style application.

3.4 Modification summary

The time-line of the components of the ZAPT software is shown in Figure 9. We started in mid-June 1992 with Bellcore's Touring Machine version 2 of May 1992. We ported that to the NeXTStep operating system in June 1992. MIT's switch module was added in September. We also received a copy of Bellcore's bridging module, which was also significantly modified to accommodate passive bridging. The final system was completed in February 1993. Other components were added to augment the TM, which included SafeNet fault management, the NeXT-style user interface startup manager, and the auto-accept SMGR and Radio components.

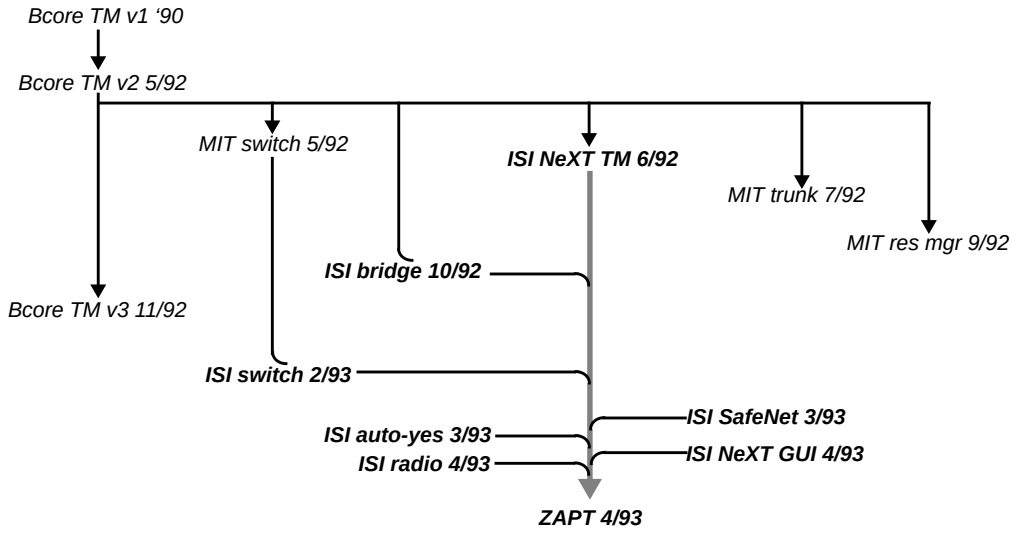

FIGURE 9. Touring Machine source tree

4.0 Accomplishments

The ZAPT project realized several accomplishments in its 1-year charter. ZAPT provides interoffice desktop teleconferencing at ARPA, with a NeXT-style application startup interface. It provides automatic additions to broadcast conferences, and automatic cross-coupling to existing Internet digital teleconferencing tools (i.e., rendezvous capability). ZAPT has been used for several Internet-wide teleconferences at ARPA in the past few months, and has demonstrated the utility of desktop teleconferencing at ARPA.

ZAPT also provides fault-tolerance via module restart, and inexpensive passive bridging. What we learned about fault tolerance and passive bridging, as well as about the TM model in general, has been shared with the research community. Our crossbar debug module has been made available on the Internet via anonymous FTP, and has been given to XN Technologies (the switch manufacturer).

Finally, the experience of developing and installing ZAPT has influenced our models of multimedia teleconferencing from another viewpoint. We have attempted to begin such a model, called "M3" - the Multicast Multipoint Model [To92]. Our model supports conference merging and subconferencing, neither of which is supported in MMCC or the Touring Machine. It also permits comingling of different data stream types, permitting, e.g., audio to go to an oscilloscope whose video goes to a monitor. MMCC and TM, as well as some proposed IETF MMUSIC Working Group models [MM93], currently enforce strict stream partitioning by type. In addition, we recommend a dynamic-state model, in which resources and state have a "free" ground state, and are kept reserved only by continual refresh requests. This permits a true fail-stop system. The physical crossbar should be part of a logical switching system that includes a logical crossbar and logical input and output gates, where session control governs the logical crossbar, and user end-control governs the gates, as noted in the general recommendations. These concepts are being developed to influence both the MMUSIC and SPT models [Sc93].

5.0 Conclusions

We found that desktop access to Internet teleconferencing is more useful than interoffice teleconferencing at the local site only. This would permit informal collaboration regardless of proximity.

Video teleconferencing has become more widespread, due to the efforts of the IETF. The MBONE now reaches over 500 hosts world-wide with multicast IP[1] [Ca92], and casual broadcast conferences are a daily occurrence. This so-called "IETF-style" teleconferencing has become popular, using Sun SPARCs, SGI Indigos, and DEC 5000s. Unfortunately, NeXTs are not among those supported, due in part to limited access to their OS sources (no multicast IP is available[2]), and differences in windowing systems (X11 isn't native). The Internet teleconferencing tools were not sufficiently mature when ZAPT began in 1992, but have become pervasive in the past year. As a result, ZAPT should eventually be replaced with the evolving Internet tools.

We also evaluated the NeXT as a general audio/video teleconferencing platform, in consideration of supporting existing Internet digital teleconferencing tools. The NeXT hardware and operating system were evaluated for IP multicasting, continuous digital audio, and continuous digital video capability. Both operating system and hardware were found to be lacking, at this time. There are other competing, proprietary digital teleconferencing systems for the NeXT hardware. Most use the SCSI Digital Eyes video digitization hardware, and provide conferencing only over a local network. The systems use Ethernet broadcast, rather than IP multicast, for data distribution. One system, called "Collaborate" (from Trident Data Systems, as a commercial product), provides audio and video, although with high latency (2 seconds). Another, called "Radio" (from CWI, freeware), provides only audio. None of the systems is capable of wide-area teleconferencing,

1. This is a measure of hosts accessing the Summer 1993 IETF.
2. We could not add IP multicast to the NeXT, due to lack of source code/tool access to the operating system.

either spanning multiple LANs or over large latencies, and both use audio and video formats not compatible with Internet video teleconferencing.

Digital packet audio and video on the NeXT were also examined as part of this effort. The Internet currently uses a suite of components that provide loose-style teleconferencing. The results of this evaluation are described in Appendix B. Also, using the Internet for regular teleconferencing requires resource reservation. There are no automatic tools for reservation of DARTnet at this time, although some are pending implementation. DARTnet currently has a manual mechanism for reservation of the switching nodes for protocol experiments, but this does not include the end-system equipment. This service should be included in any future plans to provide operational teleconferencing.

5.1 Recommendations

Based on our evaluation of the ZAPT effort, we recommend that the Touring Machine component be replaced with a teleconferencing system that supports multiple domains *a priori*. The obvious solution is to move towards existing Internet teleconferencing tools, such as MMCC and sd. These run on a number of existing platforms, but unfortunately the NeXT is not among these. The simplest solution requires replacement of the NeXT Cubes with a platform supported by the Internet teleconferencing community. If NeXTs must be supported, a separate effort would be required to port some of the call management tools (e.g., USC/ISI's MMCC, LBL's sd) to the NeXT, and to augment them to accommodate shared control of network resources, such as switches, analog-to-digital converters, etc.

5.2 Acknowledgments

We would like to thank Bell Communications Research (Bellcore), and especially Abel Weinrib, Alex Gelman, and all the participants of Bellcore's Touring Machine project for the software that was the basis of this project, and for their continued support during the project.

We would also like to thank Eve Schooler and Steve Casner of USC/ISI's MMC teleconferencing project, for support in the side-effect cross coupling, as well as helping ZAPT fit into the larger teleconferencing environment they designed at both USC/ISI and ARPA. We also thank Jon Postel of USC/ISI for his assistance.

Suzanne Woolf and Ray Bates of USC/ISI helped immensely with NeXT-specific issues, as well as with the NeXT-style user interface implementation. We would also like to thank Andrew Heybey and Mark Uhrmacher of MIT, for their switch control module, and help in debugging the hardware.

6.0 References

[Ar92] Arango, M., et. al., “Touring Machine: A Software Platform for Distributed Multimedia Applications,” submitted to 1992 IFIP Int’l. Conf. on Upper Layer Protocols, Architectures, and Applications, Vancouver, 1992.

[Ar93] Arango, M., et. al., “Touring Machine System,” Communications of the ACM, Vol. 36, No. 1, January 1993, p.69-77.

[Ca90] Casner, S., Seo, K., Edmond, W., and Topolcic, C., “N-Way Conferencing With Packet Video,” Third International Workshop on Packet Video, Morristown NJ, Mar. 1990.

[Ca92] Casner, S., and Deering, S., “First IETF Internet Audiocast,” ACM Sigcomm, July 1992. Also available as ISI Reprint Series IS/RS-92-293.

[Fr92] Frederick, R., The nv ‘network video’ tool program Unix ‘man pages. Nv is available via anonymous FTP from parcftp.parc.xerox.com.

[Ja92] Jacobson, V., The vat ‘visual audio tool’ program Unix ‘man’ pages. Vat is available via anonymous FTP from ftp.ee.lbl.gov.

[MM93] MMUSIC, The IETF MMUSIC Working Group (Multiparty Multimedia Session Control), A. Weinrib and E.M. Schooler, chairs, Proceedings of the Twenty-Seventh Internet Engineering Task Force, Amsterdam the Netherlands, July 1993, pp. 417-430.

[Sc92a] Schooler, E.M., “An Architecture for Multimedia Connection Management,” Proc. 4th IEEE ComSoc International Workshop on Multimedia Communications, Monterey CA, Apr. 1992.

[Sc92b] Schulzrinne, H., “Voice Communication Across the Internet: A Network Voice Terminal”, Dept. of Elec. Eng. Tech. Report, Univ. of Massachusetts Amherst, July 1992. Also the nevot network audio tool Unix ‘man’ pages. Nevot is available via anonymous FTP from gaia.cs.umass.edu.

[Sc93] Schooler, E.M., “Case Study: Multimedia Conference Control in a Packet-Switched Teleconferencing System,” Journal of Internetworking, Vol. 4 No. 2, June 1993, pp. 99-120.

[To92] Touch, J.D., “Multiparty Connections,” Internal Memo, USC/ISI, Marina del Rey CA, Nov. 1992.

7.0 Appendix A: automated client protocols

The following is a description of the automated client protocols developed to extend the TM to handle automated cross-coupling with external teleconferencing systems. It was not fully implemented, because its complexity began to rival that of the TM itself.

The TM is described by a set of operations - initialization, database access, call initiation (outgoing), and call indication (incoming), each of which needs to be modified to accommodate proxies. A proxy needs to register each client on whose behalf it acts, in addition to itself, via *registerClient* (Figure 10). Calls to these clients need to be translated into a call to the proxy; a “Bubble” is inserted between user clients and the central system, to effect these translations (Figure 11). Nameserver replies are filtered by the Bubble, to hide proxies from user clients (Figure 11).

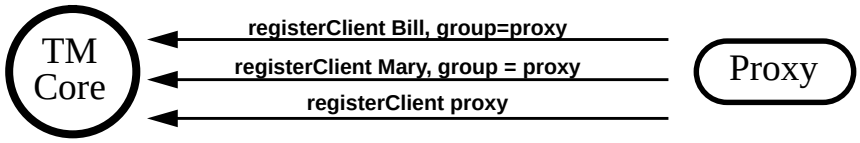

FIGURE 10. Proxy registers for each user it represents, plus itself.

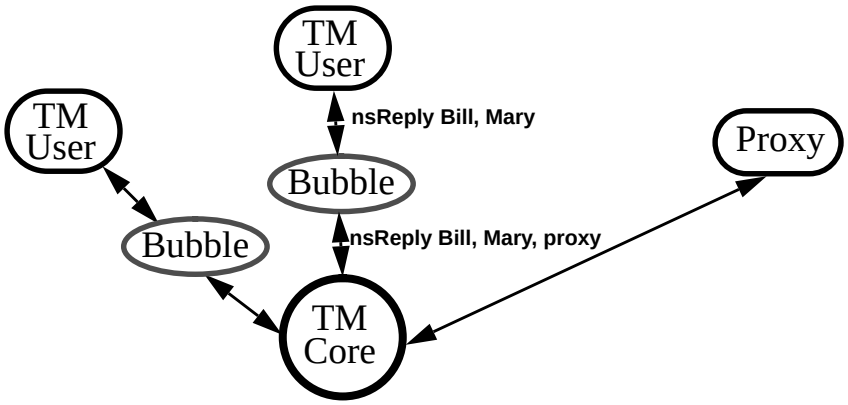

FIGURE 11. The TM general organization, with Bubble for proxy translations, with nameserver filtering.

Consider calls from the TM to the remote client (proxy call requests), and calls from the remote client to a user inside the TM (proxy call indications). TM users register via *registerClient* (**Initial**). A TM call starts with a user *sessionCreate*, acknowledged by a *sessionRequestReceived* message (**Create**). The TM core sends *sessionActionRequest* to all members, and collects a *sessionActionAccept*, *sessionActionDenied*, or *timeout* for each (**Reply**). Eventually, a *sessionActionCommit* or *sessionActionAbort* is sent to each member (including the initiator), indicating the result of the call (**Accept**). This protocol is denoted in Figure 12.

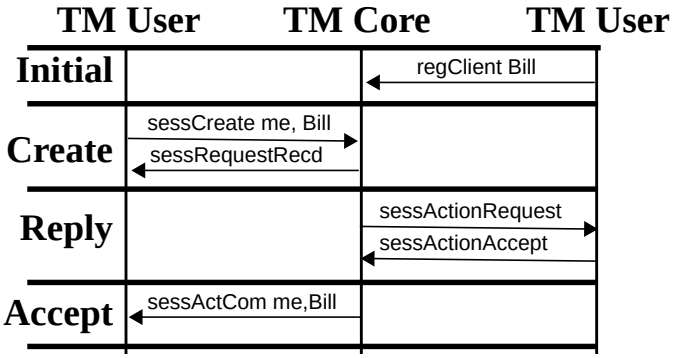

FIGURE 12. Conventional TM session establishment protocol.

When a TM user performs a call request in the augmented protocol, remote users in the initial *sessionCreate* are checked in the nameserver by the Bubble, and replaced by the indicated proxy (**Create**), which has already registered all accessible users *a priori* (**Initial**). The request is forwarded to the central system, which sends it to the proxy (**Reply**). The proxy executes an external protocol to connect to the remote proxy, and accepts or denies the request as indicated. The reply from the proxy is translated in the Bubble at the TM user, who receives the final reply (**Reply**). See Figure 13 for details.

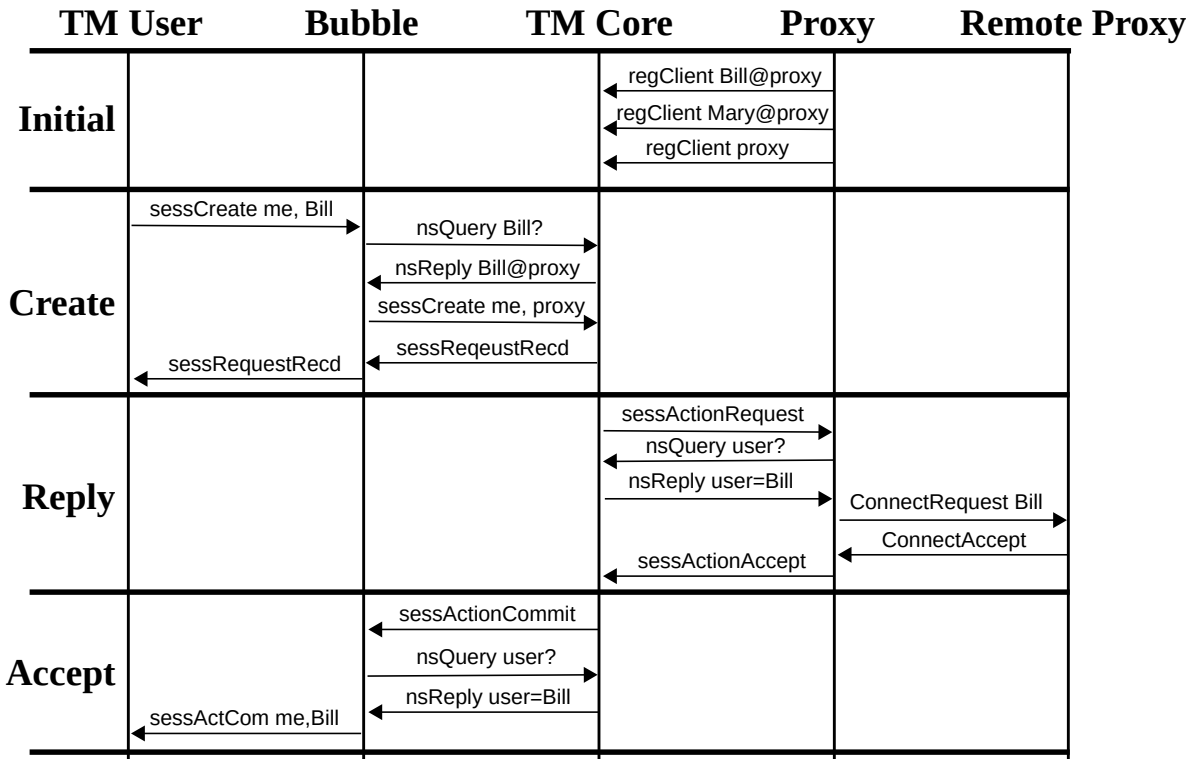

FIGURE 13. Proxy TM call request session establishment protocol.

Call indications in the augmented protocol originate at the remote proxy, by a connection request. The proxy registers the clients of the incoming call, and sends a *sessionCreate* to the central TM (**Create**). The rest of the protocol proceeds inside the TM side as before, and finally an acknowledgment is sent to the remote proxy, as in Figure 14 (**Accept**). Other actions occur as in the proxy-extensions for the call request protocol, as in Figure 13.

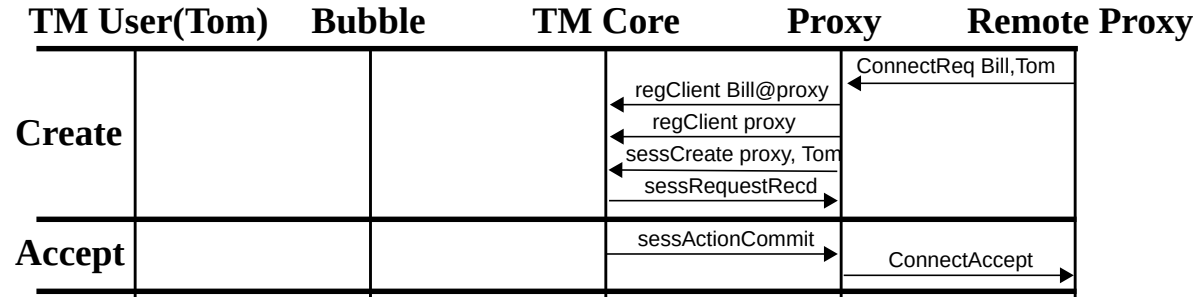

FIGURE 14. Proxy TM call indication session establishment protocol.

There is a separate protocol between the local and remote proxies, representing two-party connection establishment. The state machine and events are show in Figure 15.

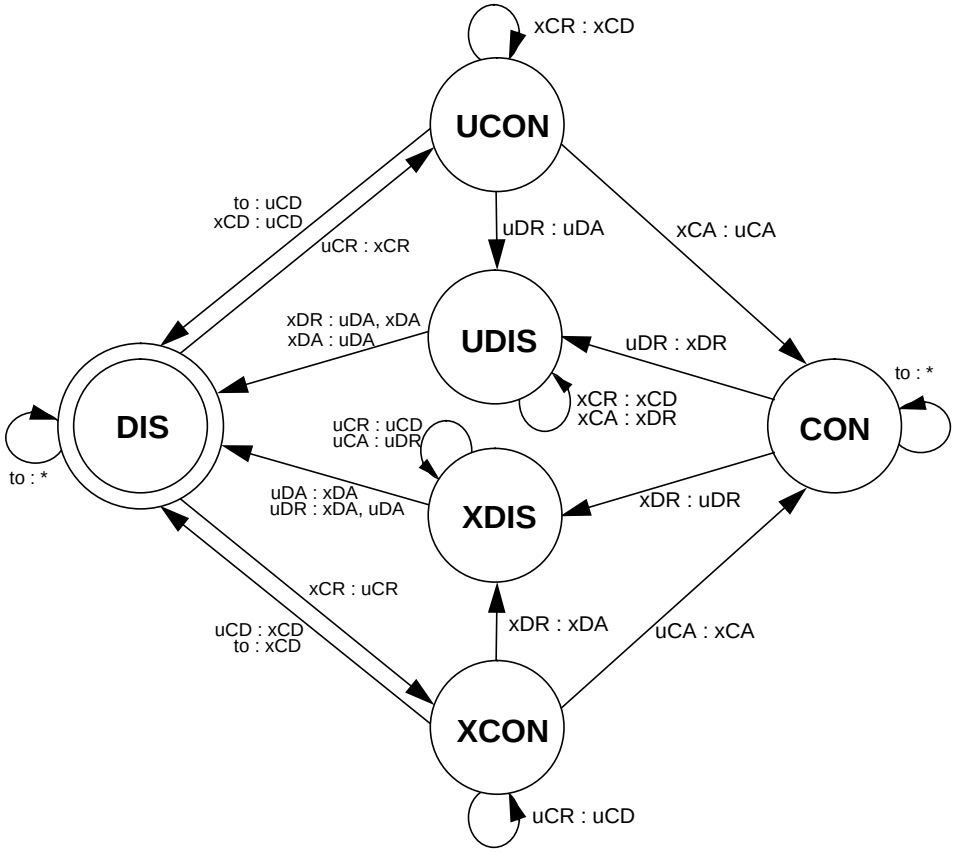

FIGURE 15. Proxy-proxy connection protocol

A word about notation in these two figures. There are 5 types of messages - connect request (*CR*), connect accept (*CA*), connect deny (*CD*), disconnect request (*DR*), and disconnect accept (*DA*). Prefixes indicate the source or sink of the message: *U* indicates user messages, *X* denotes external (remote party). For example, a user connection request is *UCR*. In the state diagram, *to* indicates a timeout, "*" is no message, and errors are not shown. The states are named connected (*CON*), disconnected (DIS), user connecting (*UCON*), user disconnecting (*UDIS*), external connecting (*XCON*), external disconnecting (*XDIS*).

We have described the differences between the TM and automated client protocol, in all other respects they should be equivalent. In particular, the timeout and fault tolerant behavior is not affected by these extensions, but would need to be implemented to provide an environment in which to effect these modifications.

7.0.1 Broadcast client protocol

We wanted to add "radio" service by extending the TM protocol. An automated TM proxy can't broadcast. Many users can auto-connect, but only one session can be active (and choice is under proxy control). Creating a session for each user receiving a broadcast is inefficient.

TM provides a broadcast mode, in which a single source is received by all users, using a "bussing" capability in the analog switch. In TM version 2, users can't join an existing session; they must be "added" by an existing member.

We modified the TM operation by creating a Radio client. All call requests are denied by Radio; such requests are assumed to be a request to join the broadcast (Figure 16). The Radio then adds the caller to its broadcast session. Radio creates the session when the first user joins, and adds users thereafter. It tears down the session when the last user leaves, releasing the proxy resources.

The Radio was designed to be cooperative with a Remote proxy client. The Remote client implemented the automated proxy protocol. Both were part of a single module, so shared state could permit either to acquire the external analog connection.

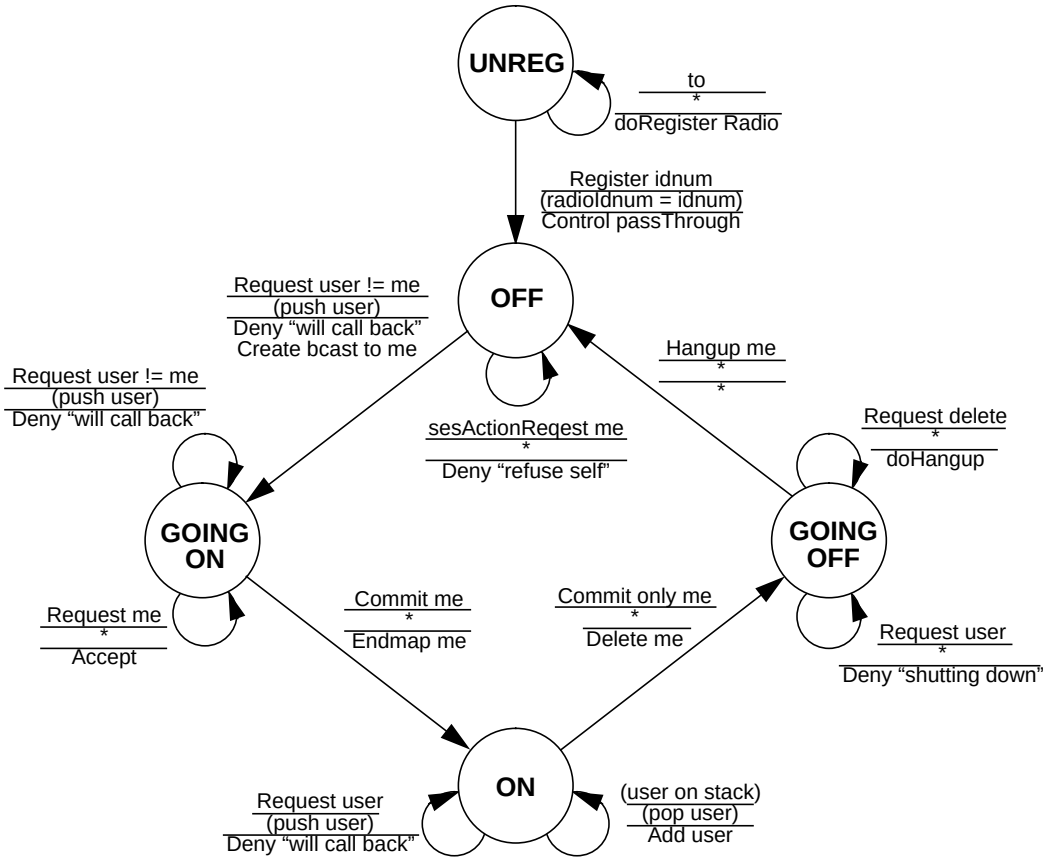

FIGURE 16. Radio client protocol (optimistic, as implemented)

Figure 16 shows a simplified Radio protocol. Each transition is labelled with received message / internal action / output message. An asterisk ("*") indicates no action or message, and *to* indicates a timeout. The "trick" shown is to use call requests as implicit join requests. The states show are unregistered, off (waiting for requests), going on, going off, and on (currently broadcasting). The protocol shown is optimistic, because an optimistic client protocol was traceable. A pessimistic protocol, with failure recovery, was not developed because of insufficient information on the TM client protocol.

We attempted to develop a more detailed Radio protocol, to better model the state transitions nd eventually provide fault tolerance. In Figure 16, ON goes to GOING ON with a user call request, which is denied, and a broadcast call initiation. GOING ON should go to another separate state when REQUEST ME is received, i.e., when the TM core asks the client to join the session it initiates.

Further detail involves the fault recovery; a partial elaboration of this state diagram appears in Figure 17. This includes refining the path of transition from OFF to ON to have 2 intermediate steps; this is inferred from the protocol messages observed with the Bubble tracer. We would prefer to have build this diagram from a TM specification of the protocol, but none is documented at this time.

At this point we can see the flaw in the general TM protocol that prohibits fault tolerance in the general case. Although we may be able to determine "abort" or timeout transitions for all other states, the ON state has no timeout possible. The TM assumes fail-stop operation which continues

a call, and state machines don't interact while ON. A fault tolerant protocol requires the ON state to have a timeout, i.e., to have a "refresh connection" message to maintain state. This kind of periodic state transition is well known in existing transport protocols (Delta-t, etc.).

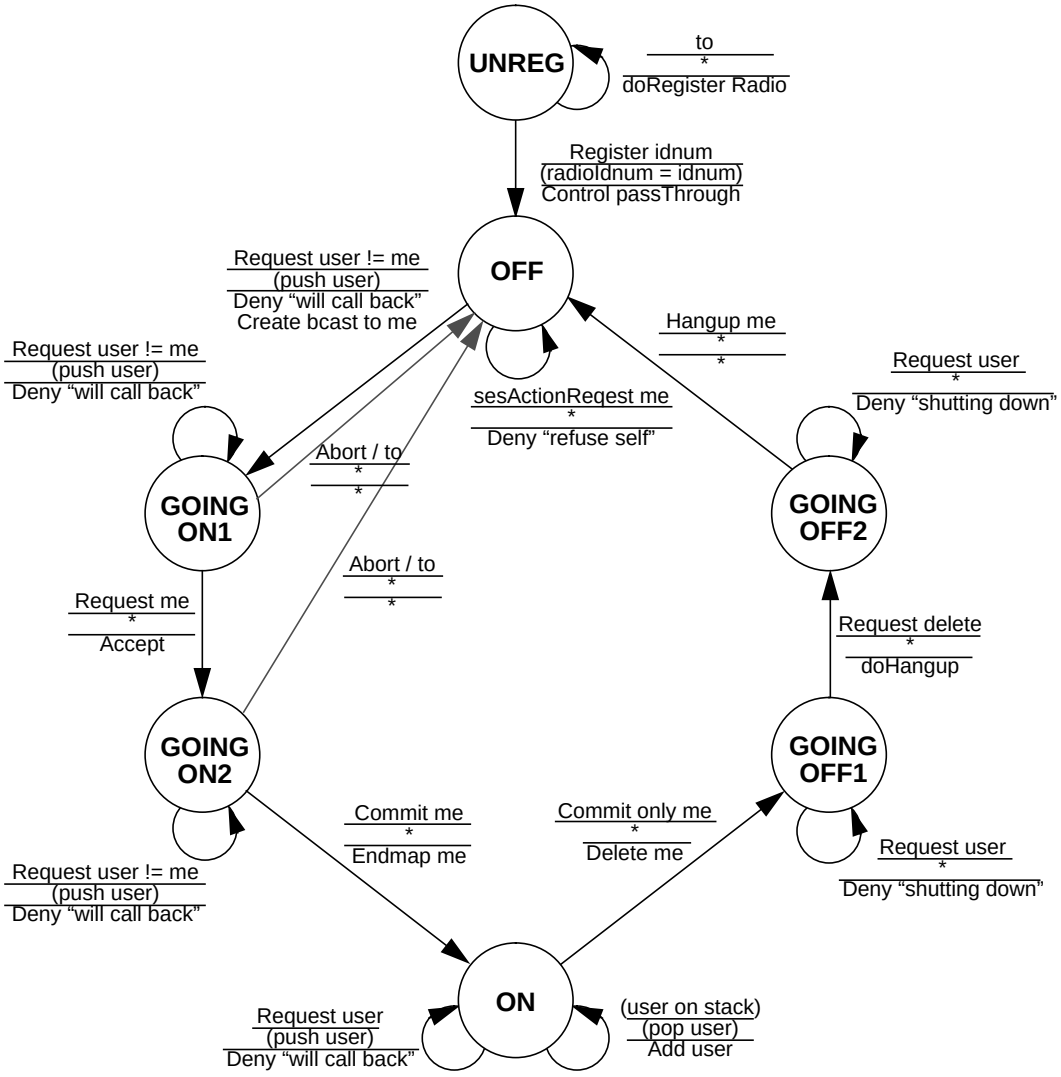

FIGURE 17. More detailed Radio client state diagram (not complete, and not implemented)

8.0 APPENDIX B: VAT / NV port results

The ZAPT system is a digital control system for analog audio and video signals, but Internet teleconferencing currently uses digital audio and video. Here we summarize our attempts to port two popular digital Internet teleconferencing tools, “vat” for audio, and “nv” for video, to the NeXT Cube. The vat audio tool port was unsuccessful, for reasons that are still not well understood. The nv video tool port was successful, although it exhibited very poor performance, and worked in receive-only mode.

8.1 “vat” port results

We had offered to port “vat”, the pervasive MBone audio teleconferencing application developed at LBL by V. Jacobson [Ja92], to the NeXT Cube[1], but have not been able to obtain sources for the past year. Porting would have been complicated by vat’s use of C++ (NeXT supports Objective C), and a window system package called “Interviews” (which is not supported on the NeXT). We also considered porting “nevot”, H. Schulzrinne’s vat-compatible application (at AT&T Bell Labs [Sc92b]), but were unsuccessful at this time, due to incomplete information on NeXT sound drivers. Nevot, as vat, and most other similar digital audio applications, uses a “/dev/audio” model of sound access. Packets of audio are read and written, and “ioctl’s” permit control of the audio device.

The NeXT treats sound differently. The NeXT plays and records sounds via system calls, which queue sound packets, and play them via DMA transfers through the DSP chip. Unfortunately, we have not been able to accomplish even trivial bidirectional continuous digital audio, and unidirectional audio has worked only with significant (1 second) latency. Use of NeXT hardware for sound would require proprietary information on the sound system, which we could not obtain.

We have been able to modify CWI’s “Radio” program (not related to the TM application we developed, also called Radio). The modified program receives vat-style packets, and plays them as they arrive. However, even on local networks with continuous packet transmission, playout is sporadic. We have not been able to correct the playout, and expect that correction would not be possible without proprietary information on the NeXT sound system.

Note that the situation is not helped by a move to NeXTStep 486. The 486 system has no sound hardware compatibility definition. The sound interface continues to be via the OS call interface, rather than via a device emulation (/dev/audio). The NeXT has an undocumented /dev/sound.

8.2 “nv” port results

We have also ported the network video program “nv”, by R. Frederick of Xerox PARC [Fr92], to the NeXT. This port was more successful than the audio port, and is usable, although very sluggish. It required porting TCL (command language toolbox) and TK (X11 window toolbox) to the NeXT under co-Xist. The resulting program works, provided the IP multicast video is shunted to the NeXT by H. Schulzrinne’s RTP “mirror” to convert it to a unicast IP stream. The sluggishness is due to the use of co-Xist for the windowing. Sample windows appear in Figures 18 and 19.

The main window does not display the “send video” options, because we were not able to port them to the NeXT. The NeXT hardware uses several different realtime video digitization boards; ours use the NeXTDimension board. NeXT does not support the board, and the effort required to perform the port would be large, and unwarranted without digital audio capability, so was not attempted. Another system, Digital Eyes, provides video digitization over SCSI interface, and others are considering its use for digital teleconferencing. The color displayed is slightly green-skewed, and apparently reflects differences in co-Xist’s 8-bit pseudocolor map vs. Sun’s.

1. *vat* currently runs on DECs, SGIs, Sun SPARCs, HPs, and i386 BSDs only.

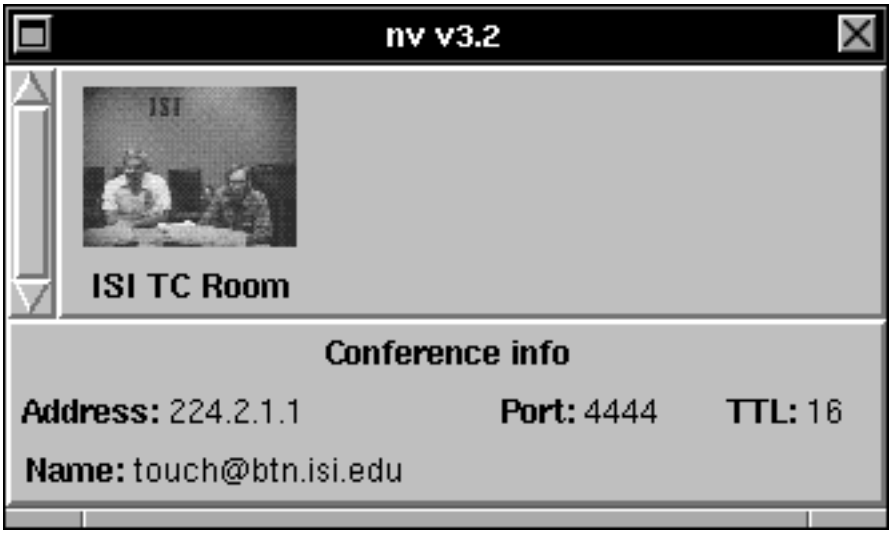

FIGURE 18. nv main window (with miniature real-time video “sample”)

FIGURE 19. Display of real-time video in a separate window (nv feature)

9.0 APPENDIX C: other tools

We developed two primary debugging tools for the ZAPT project: bubble and SoftPanel. Bubble sits between TM components, and shows the packet traffic between them. SoftPanel permits semi-intelligent control of the AutoPatch analog crossbar switch, and permits readout of the switch state.

9.1 Bubble tracer

Bubble is a TM packet tracer, and was intended to eventually evolve into the “bubble” that translated proxy operations, as described in Section 3.3.3 and Appendix A. Bubble prints out packets with a “v” prefix indicating the packet is descending to the TM core, and a “^” prefix indicating a packet is ascending out of the TM core to the user application. It also indicates the packet payload length, and attempts to print the packet payload as ASCII test. Figure 20 shows sample output of the Bubble when connected t the Radio user client.

v: 61:(endpointMap 3 “testradio:MTS:s1” 1537720159 “camera” “mic”)

^: 58:(endpointMapAccepted 3 “testradio:MTS:s1” “testradio:MTS”)

v: 97:(sessionActionDenied 915557705 “act:MTS:s10” 1537720159 “Auto-refuse by ZRADIO - will call back”)

^: 116:(sessionActionAbort 915557705 “act:MTS:s10” “testradio:MTS” “testradio:MTS: Auto-refuse by ZRADIO - will call back”)

FIGURE 20. Bubble tracer output

9.2 SoftPanel crossbar controller

SoftPanel is a software controller for the AutoPatch analog crossbar switch. It communicates via the serial port (ttya) to the switch, and send control and read status information. The ASCII-terminal display, shown in Figure 21, indicates the state of the video and audio crossbar components (video is surrounded by “v”, audio by “a”). Commands sent are shown in reverse video; commands verified are shown in regular video. It verifies the resetting of the switch, either via software reset (X-boot), or power-cycle (Zap). It also provides for sending commands, both simple (Conn, Disconn), and aggregate (Teleconf, Partclear, Fullclear). It can read the switch status (Read), and update the display (Update) independently. This software has been made available to MIT and XN Technologies, the switch manufacturer.

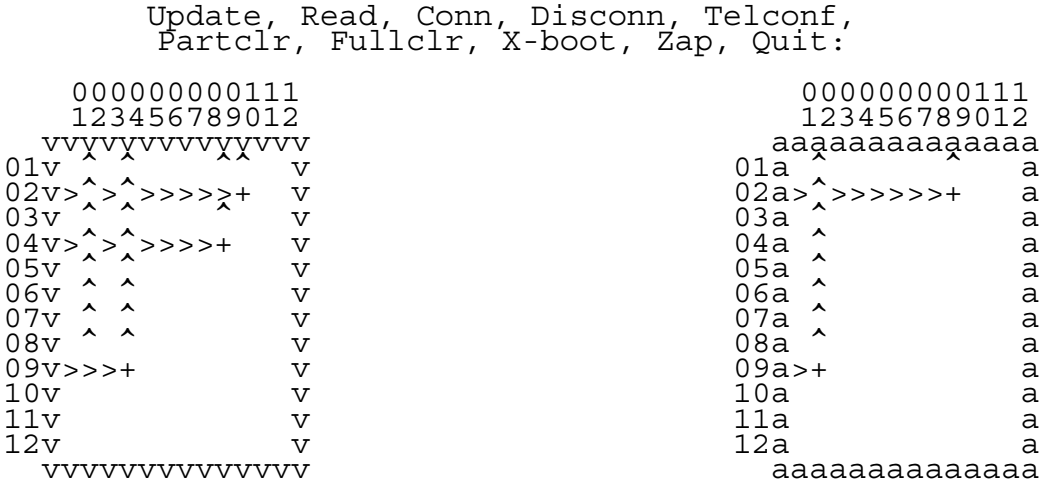

FIGURE 21. SoftPanel screen