

ENGINEERING A CAMPUS-WIDE ACCESSIBLE MUSIC LIBRARY

by

KEITH J. WINSTEIN

Bachelor of Science, Massachusetts Institute of Technology (2004)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2005

Copyright 2005 Keith J. Winstein.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly
paper and electronic copies of this thesis and to grant others the right to do so. This
work is licensed under the Creative Commons Attribution License. To view a copy
of this license, visit <http://creativecommons.org/licenses/by/2.0/> or send a letter to
Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Author
Department of Electrical Engineering and Computer Science
January 28, 2005

Certified by
Harold Abelson
Class of 1922 Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

ENGINEERING A CAMPUS-WIDE ACCESSIBLE MUSIC LIBRARY

by

KEITH J. WINSTEIN

Submitted to the Department of Electrical Engineering and Computer Science
on January 28, 2005, in partial fulfillment of the
requirements for the degree of
MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Abstract

The Library Access to Music Project has created a new kind of music library at the Massachusetts Institute of Technology. The library is always open and available in dormitory rooms and classrooms, because it transmits music on demand over the Institute's cable television system. By using the analog cable television system, LAMP differs from existing commercial offerings in that essentially any musical recording may be added to the collection — not just recordings where “digital rights” have been obtained. Additionally, LAMP is orders of magnitude less expensive than existing commercial offerings, and it is compatible with a much wider range of receiving apparatuses. With these advantages come unfortunate limitations that spring from LAMP's technical architecture and posture under copyright law. Nonetheless, LAMP has been a moderate success since its opening in October 2004, playing an average of 580 songs per day.

Thesis Supervisor: Harold Abelson

Title: Class of 1922 Professor of Computer Science and Engineering

Prologue

MONDAY, October 27, 2003, Josh Mandel and I opened the Library Access to Music Project for the first time. After two years of work, we were finally finished and flying high: several other universities had already asked how to start their own LAMP libraries, our system was swamped with listeners, and thanks to M.I.T.'s able public relations office, our faces and voices were all over the media as peacemakers in the “digital music wars.” We had figured out how to give free music to college students, and it looked like we were going to get away with it.

That evening, Josh and I were finishing up a string of television interviews, where I was boasting to one station after another about how much work we had done to make sure we had all our copyright ducks in a row. We were heading out to be interviewed on CNN when I received a call on my cell phone. “Guys, we have a major problem.” It was Ann Hammersla, M.I.T.'s intellectual property attorney.

In fact, our copyright ducks were not all in a row. Instead of purchasing 3,500 compact discs, which would have been unwieldy, we had spent \$28,000¹ to purchase two hard drives loaded with compressed copies of those 3,500 CDs from the Loudeye Corporation. It had taken more than a year of negotiations between M.I.T.'s and Loudeye's attorneys before the M.I.T. lawyers had been comfortable enough with the transaction to sign a contract. That evening, Loudeye called to say it had made a mistake. Although the company had advertised this service on its Web site and in press releases, and although our 3,500 CDs were minuscule compared with Loudeye's sales to major broadcasters,² Loudeye's general counsel told us that she had discovered — after a complaint from the Universal Music Group — that the company was not licensed to sell music files at all. Loudeye had broken the law. Every one of the 48,000 songs that the company had sent us was an illegitimate, bootleg copy.

On Tuesday, the Institute's and Loudeye's attorneys struggled to come up with a solution to avoid embarrassment for M.I.T. and legal trouble for Loudeye. This involved Loudeye's attorney

¹M.I.T. had not actually sent the check to Loudeye by the time we learned of trouble, so we are able to get our money back.

²Loudeye sold XM Satellite Radio about 120,000 compact discs' worth of MPEG audio files. Craig Johnston, *XM Radio's Music Is Massive*, RADIO WORLD NEWSPAPER, http://www.rwonline.com/reference-room/trans-2-digital/05_rwf_xm.1.shtml (accessed May 26, 2003) (on file with author). The company has also sold music to Sirius Satellite Radio.

calling the major record labels and asking for retroactive permission for the company's sale to M.I.T. Perhaps unsurprisingly, the record labels were furious at Loudeye and none too keen to help LAMP, which they saw as exploiting a copyright loophole. Universal Music and Sony Music told Loudeye that it had better find a way to get their music back, or else. Some of the other labels were friendly, but no more willing to let Loudeye off the hook. The talks went nowhere.

Wednesday, October 29, saw the heavily-promoted launch of Napster 2.0, showing off the record industry's new way of doing business. For \$10 a month, customers with Microsoft Windows computers could receive "streaming" access to a library of recordings for which the labels had been able to get "digital rights" — mostly recent releases, but still a respectable collection of about 500,000 songs. At the Napster launch party in Los Angeles, the liquor flowed late into the night, and somebody spoke with Jon Healey, a business reporter with the Los Angeles Times, about LAMP.

By Thursday, it was clear that the record labels would not retroactively grant Loudeye the permission it had needed to sell us hard drives loaded with music. M.I.T. was not itself breaking the law — the Institute had all the licenses necessary to transmit music over its cable television system. And it certainly was not our fault that Loudeye had broken the law. But this was too fine a line for M.I.T.'s lawyers to draw, especially now that the Los Angeles Times was preparing an article for the next day. Somewhere, along the way, the law had been broken. The lawyers concluded that the only sane course of action was to tell Mr. Healey that we were closing LAMP "temporarily," until we could find a legitimate source of music.

That Friday saw the public unraveling of two years of our work. Loudeye, which was privately apologetic, became publicly bellicose, suggesting that M.I.T. had screwed up its own licensing. It turned out that no company is licensed to do what Loudeye had done — selling broadcasters copies of copyrighted songs, aggregated together on a hard drive. We had shut LAMP down with no clear prospect of return.

As The New York Times put it: "It was hailed as ingenious: a way to listen to music on demand while avoiding the legal battleground of file sharing. Best of all, the music was fully licensed, so there would be no legal trouble. But it was not, and there is."

One year later, in October 2004, a more modest LAMP did eventually return. It has been open for the last three months, with a library of 1,800 CDs. So far, 811 people have used the library, playing an average of 580 songs per day. We have spent about \$34,000 from the M.I.T.-Microsoft iCampus partnership on the project — about \$15,000 on equipment, and \$19,000 on the 1,800 CDs.

To get music into LAMP without the benefit of Loudeye's hard drives, we had to purchase physical CDs and somehow make them able to be broadcast on demand. This meant "ripping" them onto a computer hard drive,³ something that is presumptively illegal unless it fits within one

³Physical jukeboxes capable of holding 2,000 CDs are generally gigantic, very expensive (more than \$100,000), and only able to play a few CDs at once.

of the copyright law's narrow exceptions. Thus the new LAMP has some unfortunate limitations — in particular, users now request music by selecting approximately six-song chunks, instead of being able to select individual tracks by themselves.

The new LAMP is now so complicated that probably no other university will make another one. The future of access to music is clearly moving in a different direction — toward Napster 2.0-like services that charge universities \$2 or \$3 per month per student, for unlimited digital on-demand streaming of 500,000 songs from the “digital rights” collection.

Still, most of our original goals have been fulfilled. Any student or faculty member at M.I.T. can now listen, on-demand, to any song in the library's collection at any time, from almost anywhere on campus. Right now, the library only holds 1,800 CDs, but any CD can be added just by purchasing it. We own and control the collection. And the whole system comes at no additional recurring cost to M.I.T. — unlike a \$3 per month service, which would cost the Institute \$380,000 a year.

Introduction

LAMP WAS BORN before file-swapping or MP3s were popular. In 1996, our high school library closed at 7:30 p.m. every night. With the school's Advanced Computing Association, or ACA, I tried to make the collection available at all hours of the day by putting it online. (We figured that if only one person was listening at a time, there should be no problem.) We ripped ten CDs onto a server named Luxo, and so we called the project the Luxo ACA Music Project, or LAMP.

At M.I.T., the Library Access to Music Project's goal was the same: to create a music library whose collection was instantly accessible, 24 hours a day. When Josh and I started in November 2001, our original plan was to transmit music over M.I.T.'s computer network. But we quickly ran into a hurdle. In America, a digital transmission of a recorded song to members of the public requires the permission of two copyright owners: the songwriter who composed the song, and the artist who sang the song.

The first group — the songwriters and their publishers — have banded together to create an easy-to-use system of collective licensing. Since before World War II, radio stations, concert halls, and nightclubs have needed only to deal with three organizations (known as ASCAP, BMI, and SESAC) in order to receive permission from the songwriters to perform and transmit essentially all copyrighted songs. This is a godsend, because it would be almost impossible for a radio station or nightclub to track down each of the hundreds of thousands of songwriters that these organizations represent.

The second group — the artists and their record labels — does not have such a system. As a result, unless a digital transmitter can fit within one of the narrow statutory exceptions,⁴ the only way to transmit songs digitally to members of the public is to negotiate permission one-by-one with the record label for each song.

For a library of 3,500 CDs or 48,000 songs, this would be almost impossible. Fortunately, radio stations, concert halls, and nightclubs do not have to do this, because they do not transmit music digitally. For historical reasons, *analog* broadcasters are only required to secure permission from the first group — the songwriters. And in fact, most universities already have licenses from ASCAP,

⁴For instance, one of these exemptions applies only to XM Satellite Radio and Sirius Satellite Radio, but not to any new competitors. 17 U.S.C. § 114(d)(2) (2000).

BMI, and SESAC. The fee is about 30 cents per student per year to each of ASCAP and BMI, and about 10 cents per student per year to SESAC. For M.I.T., this works out to about \$7,000 a year and covers the campus radio station, cable television system, and any performance with a fee for admission.⁵

Because it was simply impractical to negotiate all the licenses necessary for a digital music-on-demand system, we designed LAMP to transmit music over M.I.T.'s analog cable television system. Users log on to LAMP's Web site and select music they'd like to play, and the system plays the music over cable TV. The quality is better than FM radio and not as good as a CD. Only 16 songs can be playing at a time, but so far we have rarely bumped into this limit. Often users are content just to listen to what someone else is playing.

LAMP represents a new kind of music library — one that is open all the time and always nearby. Compared with commercial music-streaming services, LAMP is extremely inexpensive. And yet the system puts much more control in the hands of its librarian administrators, who can add any CD to the collection just by purchasing one copy. Whether the LAMP model is of enduring value, however, remains to be seen.

⁵Free performances are usually exempt under 17 U.S.C. § 110(4) (2000).

Legal Considerations

THE MOST VEXING PROBLEM facing LAMP has been, “How do we get the music into the system?” It is easy to purchase compact discs. M.I.T. already has licenses to broadcast any compact disc on demand.

The difficulty comes in trying to broadcast them automatically. A gigantic jukebox with 4,000 slots and 16 drives is simply impractical. In fact, the largest jukebox we know of (the Plasmon D2175) only holds about 2,000 CDs and 12 drives and costs about \$100,000.

An option with great appeal is simply to copy, or “rip,” compact discs onto a computer hard drive, and broadcast them from there. Unfortunately, copies are presumptively illegal under the copyright law. Even though we are licensed to *broadcast* the song, we are not licensed to *reproduce* the song.⁶ So unless we can claim an exception (with more solidity that copyright’s equitable doctrine of “fair use”⁷), we cannot make the copies.

This problem also arises in the radio industry, where broadcasters frequently would like to pre-assemble programs containing copyrighted songs. For this reason, in 1976 Congress established a privilege for “ephemeral recordings.”⁸ Essentially, the privilege says that if you are licensed to broadcast a song, then you are also licensed to make a temporary, limited copy of the song for your own internal transmission purposes. Radio stations use this provision to justify assembling large collections of “ripped” CDs in order to avoid the inconvenience of dealing with physical discs.

There are two catches, however. The temporary copies must be deleted within six months. And the statute does not really authorize copying the *song*. Instead, it allows a broadcaster to make a “transmission program,” which is in turn defined as, “a body of material that, as an aggregate, has been produced for the sole purpose of transmission to the public in sequence and as a unit.”⁹

When CDs are “ripped” the normal way, they are copied song-by-song onto a hard drive. This, M.I.T.’s lawyers decided, was not the same as making a “transmission program,” because a single song is not a “body of material” or a “sequence.”¹⁰ In response, I proposed ripping CDs in three-song

⁶In particular, permission to reproduce the song would have to come from both the songwriter and recording artist. Permission to broadcast the song by analog means is only needed from the songwriter, via ASCAP, BMI, or SESAC.

⁷17 U.S.C. § 107 (2000).

⁸17 U.S.C. § 112(a) (2000).

⁹17 U.S.C. § 101 (2000).

¹⁰This meant that most large radio stations in the country are breaking the law.

chunks, so that if a user wanted to play track 4, for instance, she could only find it in a program that also included two other tracks “in sequence and as a unit.” (For more information, see Appendix B, which reproduces this proposal.)

This was not quite good enough for M.I.T.’s lawyers, who wanted a case to validate our interpretation of the statute. How long did something have to be to qualify as a “transmission program”? Was a three-song “sequence” long enough? What about an entire CD? There are no clear answers to these questions.

Only one published judicial decision has ever interpreted the “ephemeral recordings” provision.¹¹ In *Agee*, a federal appeals court construed “transmission program” to include a 30-minute episode of the television show “Hard Copy.” But it is a stretch to say that the court’s holding means that a 30-minute string of songs necessarily qualifies as a “transmission program.”

Nonetheless, a compromise was reached between LAMP, M.I.T.’s attorneys, and M.I.T.’s vice president for research. If 30 minutes was good enough for the Second Circuit Court of Appeals, it was good enough for LAMP.

M.I.T.’s lawyers also expressed a desire that our transmission programs be the result of creative consideration. In the near future, we plan to allow users the ability to assemble their own transmission programs, and monitor what they play in order to create new transmission programs.

¹¹*Agee v. Paramount Communications, Inc.*, 59 F.3d 317 (2d Cir. 1995).

User Interface

LAMP's INTERFACE combines a Web site, where users act as disc jockeys and select programs to play, with a set of 16 campus-wide cable television channels, where users act as listeners.

Because of the requirement to transmit programs "in sequence and as a unit," the interface is quite limited. Users cannot fast-forward or rewind programs, or skip to the next song. The only options are to pause and resume, and to give up the user's claim on the channel entirely.

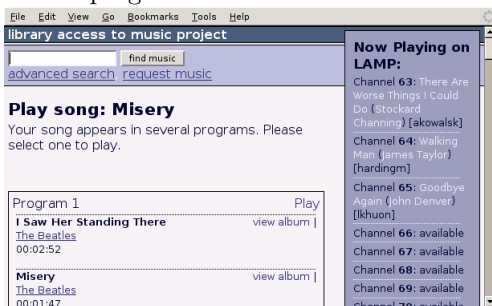
I. On the Web site, users can search for albums and songs:



II. After clicking on the album "Please Please Me," the user sees a list of tracks:



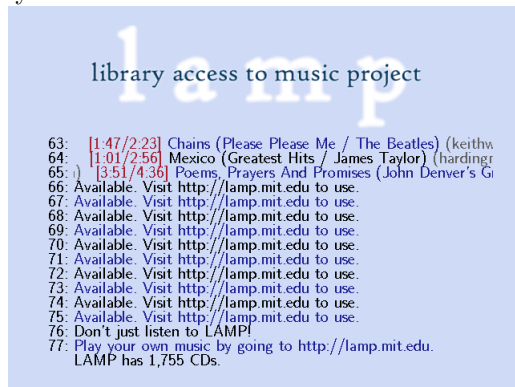
III. Clicking "play program" brings up a list of 30-minute programs that include the track:



IV. After selecting a particular program, the user is allocated a channel and the program starts playing:



V. The television screen shows what is playing on every channel at the same time:



Technical Considerations

LAMP'S TECHNICAL DESIGN comprises four major elements: the broadcast hardware, the broadcast software, the Web interface, and the CD-ripping system.

Broadcast Hardware

The broadcast hardware includes a rack-mount 1.8 GHz Pentium 4 computer with 16 sound outputs and one video output.¹² Each sound output, along with the common video output, is connected to a Blonder Tongue AMCM-806 rack-mount NTSC modulator. These all plug into a 16-way combiner and out to M.I.T. Cable's own combiners and amplifiers.

The transmission programs themselves are stored in MP3 format on an array of four 500-gigabyte FireWire hard drives.



Winstein and Mandel with the broadcast hardware.

Broadcast Software

The software running on the rack-mount computer includes two custom C++ programs running on GNU/Linux. One is a 16-way single-threaded MP3 decoder, based on the MAD library. The other uses X to render the video interface, synchronizing with the VGA vertical retrace by using the `inb(0x3da)` method. These programs run with real-time priority and locked memory, and are controlled via an ad hoc protocol over Unix domain sockets. The source code of the two programs can be found in Appendix C.

Web Interface

A separate Intel PC, running GNU/Linux and the Apache Web server, runs the Web interface, which is written in PHP and uses the MySQL database. The PHP CGI scripts communicate with the broadcast software by using a series of “glue” programs written in Perl. These programs take command-line arguments to construct messages (e.g., “play program 12”) in the ad-hoc Unix domain

¹²The computer's VGA video output is converted to NTSC by a Focus TView Gold scan converter, with which we have been somewhat unhappy. It is very difficult to produce smoothly scrolling text, because there is no way to lock the VGA output's vertical retrace to the TView Gold's internal 59.94 Hz clock. Thus even with the VGA clock set to as close to 59.94 Hz as we could muster, our text jumps once every 38 seconds.

socket protocol in order to communicate with the broadcast software. Packets are carried over an encrypted SSH tunnel to the broadcast hardware PC.

CD-ripping System

The CD-ripping system includes ten Sony CDP-CX455 400-CD consumer CD changers, which sell for about \$210 each, as well as ten RBX1600 audio hubs, which were donated by Streetfire Sound Labs and normally sell for \$700 each. The RBX1600 is an embedded-Linux computer with an ARM xScale processor and the ability to control four of Sony's CD jukeboxes at a time as well as listen to the output from any one of them.

When we received a prototype RBX1600 for development, it had the ability to control jukeboxes, receive their audio (over an S/PDIF optical digital connection), and route the audio from one of them to its S/PDIF and analog coaxial outputs.¹³ But the software on the RBX1600 did not have access to the bits being received. To remedy this, I wrote a Linux device driver for the xScale's I²S sound device to allow us to have access to the received bits in software. The source code is listed in Appendix C.



The ten CD changers, ten RBX1600s, Web server, and MP3 encoder.

In order to access the CDs, I wrote a Web server in C that runs on the RBX1600 and whose file store is the CD digital audio in the jukebox connected. For instance, if a Web browser requests the URL <http://rbx1600/29/5>, the Web server will reply with the digital audio on CD 29, track 5. The Web browser can also visit a URL such as <http://rbx1600/29/trackcount>, in order to learn how many tracks are on CD 29.

Development of the Web server required significant debugging and defensive programming in order to account for and work around the various corner cases of the Sony jukebox protocol and implementation. A source code listing is in Appendix C.

The ten RBX1600s send uncompressed CD audio over a 100 Mbps Ethernet to a GNU/Linux Intel PC, which encodes the ten streams into MP3 in real time using the gogo MP3 encoder, which is based on LAME. The output of gogo is piped over an SSH connection to the broadcast equipment's hard drives — no temporary copy of any substantial length is ever saved before the actual ephemeral recording.

One of the most work-intensive parts of CD-ripping is opening, cataloging, and loading all of the CDs. This process is greatly aided by use of a \$60 bar-code scanner, but even so, with three people to open, scan, and load CDs, it still takes about 25 seconds per CD. Loading 1,800 CDs took about 13 hours of elapsed time, or about 40 man-hours.

¹³Although the RBX1600 can control four jukeboxes and listen to any one of them, it can only listen to one at a time. Therefore, ten RBX1600s were necessary to be able to rip from ten CD jukeboxes simultaneously.

Conclusions

LAMP re-opened in a beta-test phase on October 25, 2004, and since then it has been a moderate success. So far 811 people have used the library to play an average of 580 songs per day, or 77 programs per day.

The most popular albums have been:

Number of Users	Album
28	Masters of Classical Music (Delta)
26	A Rush of Blood to the Head (Coldplay)
25	International Superhits! (Green Day)
24	Disney's Princess Collection (Walt Disney)
23	The Best of 1990-2000 (U2)
22	The Best of 1980-1990 (U2)
19	Greatest Hits 1970-2002 (Elton John)
17	Californication (Red Hot Chili Peppers)
17	Sgt. Pepper's Lonely Hearts Club Band (Beatles)
17	Greatest Hits (Tom Petty and the Heartbreakers)
17	The Phantom of the Opera (Andrew Lloyd Weber)
17	Bob Dylan's Greatest Hits

The fact that anyone can see what anyone else is playing produces a social aspect to the library. One user recently donated his entire CD collection to LAMP, so that he could share his music with everybody else at M.I.T.

The library is also an interesting way to find others with similar musical tastes. The following is an e-mail received by the author after playing Dvorak's Seventh Symphony on the LAMP prototype in 2002:

Dear Keith,

I just happened to be up at three in the morning because I'm pathetic and don't have a life, and I was flipping through the channels and saw that you had selected a beautiful symphony piece. Obviously, you're a

man of great taste and intelligence. I'd like to get to know you better. E-mail me if you're interested.

Sex depraved freshman.

P.S. Could you send me a photo of yourself?

Clearly, some users may be uncomfortable receiving e-mails like the above as a consequence of using LAMP. The social etiquette to using a public, campus-wide music library has yet to be worked out.

It has been an exciting ride, and I have been very proud to bring better access to music to the M.I.T. community. One of the questions I asked in my May 2003 Advanced Undergraduate Project — “whether users at MIT will actually want to listen to music over their televisions” — can be answered with a qualified yes.

On the other hand, a realistic assessment will probably conclude that the system is too complicated, resource-hungry (it takes 16 cable TV channels), and confusing for it to be likely that another university will implement its own LAMP. Whether the benefits of saving \$300,000 a year and having control over one's own collection can outweigh these difficulties is not yet clear. We will have to see.

Acknowledgments

This work would not have been possible without a large number of people. I am very fortunate to have worked with Josh Mandel (who did not know he was signing up for a three-year project) on every part of LAMP. Hal Abelson has been a fantastic adviser to both of us, especially when counseling caution. Randy Winchester and Jon Ward of M.I.T. Cable have been extremely generous with their time and resources.

Without the contributions of Caroline Niziolek, Austin Roach, John Hawkinson, Waseem Daher, and Mary Ross, LAMP would have been much the worse. And without Luis Loya, Caitlin Murray, and Ken Takusagawa, LAMP would have happened much, much later. Thanks for scanning in all those CDs, guys.

LAMP probably would have opened much earlier — and closed for good right after — were it not for Ann Hammersla and Mark Fischer, M.I.T.'s in-house and outside copyright attorneys. Their incredible patience throughout the last three years is amazing. They've never told me what M.I.T.'s legal bill has been through all this, but I'm pretty sure it's a lot. It is remarkable that the Institute was willing to spend so much time and money to indulge two undergraduates' troublesome project instead of just saying "no, too risky." I am also grateful to Professor Kenneth Crews of the Indiana University School of Law, and to Professors William Fisher and Jonathan Zittrain of Harvard Law School, all of whom provided invaluable suggestions on the legal front.

Of course, LAMP never would have been possible without the support of M.I.T. iCampus, and in particular Becky Bisbee, Jessica Strong, Selene Victor, and David Mitchell. I am sincerely grateful to Microsoft Research, which funds iCampus. I know I rag on you guys a lot, so thank you doubly to Microsoft for supporting M.I.T. and LAMP.

Without a generous donation of time and equipment from Mark Matossian and Stephen Street of Streetfire Sound Labs, LAMP would never have been able to re-open. The same is true of the support we received from Jonathon Weiss, M.I.T. Information Services and Technology, and the Student Information Processing Board.

Finally, thank you to Bill McCloskey, Drew Massey, Don Schmidt, Matt Wicks, and to my parents.

Appendix A: Engineering an Accessible Music Library: Technical and Legal Challenges

Keith J. Winstein (keithw@mit.edu)

May 26, 2003

1 Introduction

Digital music has provoked a shift in the way music is sold. Instead of purchasing a compact disc recording, consumers may now copy one illegally from a stranger via file-trading networks that are themselves barely legal.¹ As a result, the major music distributors have invested millions of dollars in combating illicit distribution by offering their own services to sell music to consumers digitally and instantly.²

But digital music has not provoked a change in the way music is traditionally shared, that is, the technology of music libraries and radio stations. This is partly because it is not clear how libraries and stations can take advantage of the benefits of digitization under existing copyright law. Public libraries still distribute physical compact discs, audio cassettes, and vinyl records. With limited exceptions for material used in music classes, university libraries are in the same boat. Indeed, the only reason libraries are allowed to share audio recordings at all is because of a special exemption in the copyright law: the sharing or renting of audio recordings by for-profit entities is forbidden.³ Similarly, the only radio stations to have taken advantage of new ways of broadcasting made possible by digital technology, XM Radio and Sirius Radio, were provided with a special exemption in the Copyright Act that is no longer available to newcomers,⁴ and even so XM appears to have exposed itself to possible copyright infringement liability on a massive scale.⁵

In this paper, I outline a new distribution scheme appropriate for university and public libraries that provides significant benefits in accessibility of music to patrons. This system—a combination of a library, a radio station, and a

¹MGM Studios, Inc. v. Grokster, Ltd., No. CV-01-8541, 2003 U.S. Dist. LEXIS 6994 (C.D. Cal. April 25, 2003) (“Grokster” and “Fasttrack” services not contributory infringers); *In re Aimster Copyright Litig.*, No. 01-c-8933, 2002 U.S. Dist. LEXIS 17054 (N.D. Ill. Sept. 4, 2002) (“Aimster” or “Madster” service found likely contributory and vicarious infringer); *A&M Records v. Napster, Inc.*, 239 F.3d 1004 (9th Cir. 2000) (“Napster” service infringes).

²Universal Music and Sony have launched Pressplay, since sold to Roxio, the company that bought the Napster name. EMI, Bertelsmann, and Warner Music founded a similar service, MusicNet. And Apple Computer launched the iTunes Music Store earlier this year. Amy Harmon, *Deal May Raise Napster From Online Ashes*, N.Y. TIMES, May 19, 2003, at C1. These services operate under license from the major labels and music publishers and provide music with varying degrees of software restrictions.

³17 U.S.C. § 109(b)(1)(A) (2000). This produces the amusing situation where it is legal for a business to rent out a book, and legal to rent out a video recording of someone reading the book, but not legal to rent out an audio recording of someone reading the book.

⁴Exemption for “preexisting satellite digital audio radio service[s],” 17 U.S.C. § 114(d)(2) (2000).

⁵See section 4.2 below.

jukebox—is made possible by digital music compression and the Internet, but at the same time rests critically on analog distribution. I discuss the hurdles implementing this system at the Massachusetts Institute of Technology, where I expect to launch our new digital music library, known as the Library Access to Music Project, by this fall.

The goals for LAMP were to provide on-demand access to a wide range of music to students at MIT. The system, now in beta testing, allows students and faculty to “check out” a channel on the MIT cable television system for their exclusive use for 80 minutes. During that period, they may request, via a Web site, any CD or individual track from the system’s repository, and it will play over the TV channel allocated to the user. (Additionally, a simple video display shows the title of music playing on each of the channels allocated to the project and the name of the user controlling the channel.⁶) Anyone may listen to the music, but only the user controlling the channel may fast-forward, rewind, skip tracks, or select additional music. With the resources MIT Cable has allocated for the project, 16 channels may each be individually controlled by a different user at a time.

A variety of analogies describe the project: a music library with 16 seats, an array of 16 jukeboxes that anyone may listen to across campus, a pool of CDs shared among dormitory residents, a music library where you can call up the librarian and ask to have a CD played into the telephone, a city with 16 radio stations, each taking requests. But the particular architecture chosen for LAMP was motivated by technical and legal challenges in three areas: playing the music, responding to requests for music-on-demand, and loading music on the system. I discuss each in turn.

2 Playing Music

2.1 Technical Challenges

The technical requirement for LAMP was to be able to play audio of as high fidelity as possible over the MIT cable television system on 16 channels at once. With the assistance of MIT Cable, we purchased a PC-compatible 1.8 GHz Pentium 4 rackmount server⁷ with two M Audio Delta 1010 sound cards.⁸ Each provides eight channels of 24 bit 96 kHz audio output. The outputs from each sound card feed into 16 Blonder Tongue AMCM-806 rackmount modulators,⁹ and into a 16-channel combiner¹⁰ and finally into amplifiers for the MIT cable system.

⁶See *Privacy*, below.

⁷\$3,900 total, including the sound cards.

⁸\$1,000 each at purchase.

⁹\$240 each. The modulators support stereo audio input using the FCC-recommended EIA TVSB No. 5 specification, but do not produce stereo audio from separate channels by themselves.

¹⁰\$190.

On the software end, we needed to be able to saturate 16 channels at once. We chose the Ogg Vorbis audio compression system because it is believed to be unencumbered by patents and is designed by an MIT alumnus and acquaintance.¹¹ Using the provided `vorbisfile` API, I implemented a single-threaded 16-channel stereo decoder that outputs monophonic audio. When all 16 channels are in use, the system CPU is approximately 80% non-idle (including the video display, discussed below).

The system's quality is acceptable but not stellar. We have measured total signal-to-noise ratio, on a cheap television, at approximately 45 dB, or between 7 and 8 bits of resolution. Bandwidth extends from about 30 Hz to 13 kHz. We are optimistic that fancier FM-reception hardware, such as in a modern VCR, will produce better results. We have not yet begun to implement the stereo standard (including noise reduction), which requires either \$200-per-channel stereo encoders or implementing the EIA specification in software (and using the 96 kHz capability of the sound cards).

2.2 Legal Challenges

Does playing copyrighted music over the MIT cable television system require the permission of the copyright owner? To answer this question, we must first note that there are generally two copyrights in a piece of recorded music. There is the copyright on the underlying *composition*, known as a “musical work.” And there is a separate copyright on the *sound recording*.

For instance, Jesse Harris won the 2003 Grammy award for song of the year for writing the song “Don’t Know Why.” Under the federal Copyright Act, he (or anybody he has assigned the copyright to; in this case Sony/ATV Songs) owns the copyright on this “musical work” as soon as he writes it. And in general, the copyright owner’s permission is necessary to perform any of the so-called “exclusive rights” of the copyright owner, enumerated in 17 U.S.C. § 106 (2000):

Subject to sections 107 through 122, the owner of a copyright under this title has the exclusive rights to do and to authorize any of the following:

1. to reproduce the copyrighted work in copies or phonorecords;
2. to prepare derivative works based upon the copyrighted work;
3. to distribute copies or phonorecords of the copyrighted work to the public by sale or other transfer of ownership, or by rental, lease, or lending;
4. in the case of literary, musical, dramatic, and choreographic works, pantomimes, and motion pictures and other audiovisual works, to perform the copyrighted work publicly;
5. in the case of literary, musical, dramatic, and choreographic works, pantomimes, and and pictorial, graphic, or sculptural works, including the individual images of a motion picture or other audiovisual work, to display the copyrighted work publicly; and

¹¹Christopher M. Montgomery '94.

6. in the case of sound recordings, to perform the copyrighted work publicly by means of a digital audio transmission.

So, for instance, to reproduce “Don’t Know Why” in copies or “phonorecords” (e.g. compact discs), you need Jesse Harris’ permission. And to “perform the copyrighted work publicly,” you also need his permission. Would playing “Don’t Know Why” over the MIT cable television system without his permission infringe this right to perform “publicly”? Although it is not crystal-clear that it would,¹² transmitting a song over MIT Cable probably does constitute a public performance.

Fortunately, for purposes of licensing public performances in America, virtually all songwriters and composers have delegated the power to license these performances to one of three organizations: the American Society of Composers, Authors & Publishers (ASCAP), Broadcast Music, Inc. (BMI), and SESAC, Inc., formerly the Society of European Stage Authors and Composers. All three are in New York City within four blocks of Columbus Circle, and all three are happy to issue blanket licenses to universities. In fact, most universities already have licenses from all three organizations.¹³

MIT now pays 5.25 cents per student per year for blanket cable television rights from ASCAP and BMI. SESAC includes on-campus cable television rights in their standard university license. The total price for performance rights paid by MIT is about \$4,000 a year.

But this is not the only copyright to contend with. There is also the separate copyright in each *recording* of Jesse Harris’ song. The most popular recording of this song, by singer Norah Jones, won the 2003 Grammy for best female pop vocal. The copyright on the recording is owned by Norah Jones or anybody she has assigned it to: in this case, the Blue Note Records division of EMI Music.

Do we need EMI’s permission to play “Don’t Know Why” over MIT Cable? Here the critical language is subparagraph 6 of § 106: “in the case of sound recordings, to perform the copyrighted work publicly *by means of a digital audio transmission*” (emphasis added). As long as our transmission (over MIT cable) is analog (like most television), we do not need Norah Jones’ or EMI’s permission to play her song, nor do we need their permission to play the song over the radio. If we were to transmit the music to students digitally instead (e.g., over MIT’s computer network), we would have to seek permission (which we believe would be excruciatingly difficult to obtain for most songs, with no acceptable bulk-licensing procedure available) or try to obtain a “mandatory” license under 17 U.S.C. § 114 (2000), which forbids “interactive” or on-demand services.

This is the critical reason LAMP has a hybrid structure: it is controlled over the Internet, but the actual audio is played over cable television.

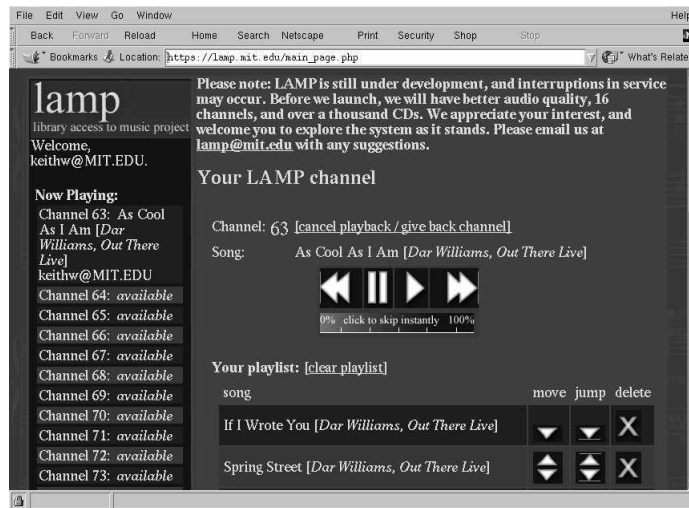
¹²The question for purposes of the transmit clause is at least partly whether the relationship between MIT and its students is a “commercial, ‘public’ one.” *On Command Video Corp. v. Columbia Pictures Indus., Inc.*, 764 F. Supp. 1372 (N.D. Cal. 1991).

¹³Note that university broadcast radio stations receive a special government-set price, currently \$578 per year, in the copyright regulations, at 37 C.F.R. § 253.5. Most performances by university orchestras and bands do not require a license because of the exception for in-person free performances by unpaid performers at 17 U.S.C. § 110(4)(A) (2000).

3 Music on Demand

Having resolved the question of playing audio over MIT cable, few additional conceptual challenges are presented by the requirement to play it on demand.

My partner Joshua Mandel implemented, using the PHP Web-programming language, an online queueing system to schedule music on each of the 16 channels made available by MIT cable. It communicates (pause, play, stop, skip, etc.) with the 16-channel Vorbis decoder, discussed above, using an ad hoc protocol over Unix domain sockets. Students and faculty visit <http://lamp.mit.edu>, present their MIT-issued SSL certificates (to prevent outsiders from hogging a system they wouldn't be able to listen to anyway, and to keep a one-channel-per-student rule in place), and are allowed to enqueue songs similar to any MP3 player:



Meanwhile, the television is displaying a status display listing each of the channels and what song, album, and user are on each, along with the time position in the song and the amount of time in the user's 80-minute allocation. I implemented the display (also signalled via Unix-domain sockets) in C++ and raw Xlib.

l a m p

library access to music project

63: v. 32m) [1:34/5:12] If I Wrote You [Dar Williams, Out There L
64: [2:19/4:09] Iowa [John Linnell, State Songs] (jmandel, 50m)
65: Available. Visit <http://lamp.mit.edu> to use.
66: Available. Visit <http://lamp.mit.edu> to use.
67: [2:10/3:05] Rhapsody in Blue [George Gershwin] (hal, 12m)
68: Available. Visit <http://lamp.mit.edu> to use.
69: Available. Visit <http://lamp.mit.edu> to use.
70: Available. Visit <http://lamp.mit.edu> to use.
71: Available. Visit <http://lamp.mit.edu> to use.
72: Available. Visit <http://lamp.mit.edu> to use.
73: Available. Visit <http://lamp.mit.edu> to use.
74: Available. Visit <http://lamp.mit.edu> to use.
75: Available. Visit <http://lamp.mit.edu> to use.
76: Available. Visit <http://lamp.mit.edu> to use.
77: Available. Visit <http://lamp.mit.edu> to use.
LAMP is in beta testing.

The display is typeset in Computer Modern Sans Serif, part of the standard \TeX suite of fonts designed by Donald Knuth. We use the Adobe Type 1 version of the font produced by Blue Sky \TeX Systems.

The display scrolls right-to-left for song titles too long to fit in one screen. (Every line scrolls one revolution in eight seconds. We found this far preferable to a constant speed for each line.) The display is synchronized to the NTSC vertical retrace, producing smooth motion with no retrace lines, using two methods.

First, the Xlib client is synchronized to the video card's retrace using a nonportable method: reading the raw I/O port corresponding to the monitor's retrace. This requires going into a special non-protected access mode under Linux.

```
if ( realtime ) {
  myparams.sched_priority = sched_get_priority_max( SCHED_FIFO );
  if ( sched_setscheduler( 0, SCHED_FIFO, &myparams ) == -1 ) {
    perror( "setscheduler" );
    exit( 1 );
  }
  if ( mlockall( MCL_CURRENT | MCL_FUTURE ) != 0 ) {
    perror( "mlockall" );
    exit( 1 );
  }
  rtc = open( "/dev/rtc", O_RDONLY );
  if ( rtc == -1 ) {
    perror( "/dev/rtc" );
    exit( errno );
  }
  ioctl( rtc, RTC_IRQP_SET, 8192 );
  ioctl( rtc, RTC_PIE_ON, 0 );
  if ( iopl( 3 ) != 0 ) {
```

```

    perror( "iopl" );
    exit( 1 );
}
}

```

And later, to synchronize with the retrace, we busy-wait on that I/O port (after sleeping to approximately the right place in time):

```

if ( realtime ) {
    while( 1 ) {
        if( (inb( 0x3da ) & 8) ) break;
    }
    while( 1 ) {
        if( !(inb( 0x3da ) & 8) ) break;
    }
}
}

```

4 Loading Music

Having resolved the problems inherent in playing music to users and allowing them to select that music, there remains the significant problem of loading that music onto our system so as to be playable on demand. It is this problem that has held up the actual launch of LAMP at MIT since November 2002.

4.1 Technical Challenges

There are two ways to load music onto the LAMP server: by purchasing compact discs and copying their contents, compressed with Ogg Vorbis, onto the hard disc (“ripping” them), or by purchasing recorded music already “ripped,” compressed, and aggregated onto a hard drive.

The first method is advantageous because MIT already owns 7,000 CDs in its music library, and we could rip these for little expense. (With \$1,500, we have constructed a ripping station that can rip and compress 200 CDs with about 2 hours of work and five days of waiting.)

However, this is much less convenient than sending a list of CDs to a wholesaler and receiving, the next week, a hard disc containing those CDs already compressed and ripped. As far as we are aware, there is only one company in the country, Loudeye Corp. of California, that offers this service. This is the company that XM Radio used to build a hard disc array containing 120,000 CDs.

A third method, of keeping an actual CD jukebox connected to the LAMP broadcast equipment with hundreds or thousands of physical CDs in it, we have concluded is likely not practical for reliability, space, and financial reasons.

There remains the problem of how to decide which CDs to purchase. We ran a three-week online survey that collected about 2,000 suggested albums from dormitory residents. We plan to supplement these with the complete works of various classical and romantic composers.

4.2 Legal Challenges

The legal challenges of loading music onto the system are much more difficult. Method one, ripping, involves making systematic reproductions of thousands of compact discs. This necessarily implicates the exclusive reproduction right (17 U.S.C. § 106(1) (2000)), and we must look to exemptions.

The only exemption that is clearly applicable is known as the “ephemeral recording” exemption, 17 U.S.C. § 112 (2000).¹⁴ The gist of the exemption is that a party with permission to *perform* or broadcast a copyrighted work (such as a radio station with an ASCAP/BMI/SESAC license) may make a so-called “ephemeral” recording of the work for the purpose of aiding the licensed performance or broadcast.

Using this exemption, most commercial radio stations in the country now use tools such as “Magic,” “AudioVault,” and “Prophet” to rip the CDs they plan on playing for scheduled broadcast later.¹⁵ Our own ripping appears to be covered under the exemption as well. But under the actual text of the section 112, it is not clear that either we or the radio stations would be following the law.

In particular, the provision allows licensed broadcasters to make “no more than one copy or phonorecord of a particular transmission program embodying the performance.” A “transmission program,” in turn, is “a body of material that, as an aggregate, has been produced for the sole purpose of transmission to the public in sequence and as a unit.”¹⁶

Although our song-by-song on-demand copies would be identical under the law to those made by most commercial radio stations, MIT is not confident that these constitute copies of an “aggregate” “transmission program.”¹⁷ This appears to be another quirk in the statute: did Congress really intend to make every radio station be breaking the law by copying songs internally before broadcasting them? Nonetheless, MIT is not willing to depend on this exemption.

¹⁴There are actually two relevant exemptions: the exemption for licensed performances in subsection (a), and the exemption for nonprofit institutions in subsection (b). Subsection (b) is more favorable because it would allow us to retain ripped copies for seven years instead of just six months. But on its terms it does not give permission to copy musical works even though we have a performance license. We would still have to obey the six-month limitation when the musical work is in copyright, even though we have licenses to broadcast the musical work. Was it really Congress’s intent to exclude the presence of a performance license as a qualifying reason for subsection (b), and thus effectively legislate a six-month limit for universities making ephemeral recordings of non-classical music where the musical work is still in copyright? This would not be consistent with Congress’ stated intent in the legislative history for subsection (b), but it does appear to be the law.

¹⁵Most radio stations do not appear to follow the six-month limit.

¹⁶17 U.S.C. § 101 (2000).

¹⁷The legislative history is unclear. See H.R. Rep. No. 105-796 (1998) (accompanying the Digital Millennium Copyright Act, Pub. L. No. 105-304, 112 Stat. 2860 (1998)), which says that the purpose of the DMCA’s 112(e) amendment is to allow webcasters “to reproduce multiple copies of a sound recording,” and that webcasters would need this amendment because “Under section 112(a), as amended by this bill [the DMCA], a webcaster with a section 114(f) statutory license is entitled to make *only a single copy of the sound recording*” (emphasis added) (no mention of “transmission program”).

This leaves only the option of buying the music already in digital format. There is one company in the country, Loudeye, that we are aware of that provides this service under license from the music labels.¹⁸ Loudeye, however, does not have permission from the copyright owners in the musical works it sells. It was until recently Loudeye's position that the permission of songwriters was not necessary for them to sell us recordings. Since the songwriters and music publishers have sued over a very similar situation,¹⁹ we were not willing to purchase recordings from Loudeye until we could receive permission from the songwriters and music publishers.

Unfortunately, the organizations that represent virtually all of those songwriters and publishers, the National Music Publishers Association and its licensing arm, the Harry Fox Agency, have never before been asked to approve this kind of transaction.²⁰ Five months after first receiving our request for a license to buy these CDs (on a hard disc) from Loudeye, the Harry Fox Agency concluded that no license was necessary. Four hours later, Harry Fox's "New Media Coordinator" called me back to say they had changed their mind and decided Loudeye *did* need a license from them.

In fact, it appears that when XM Radio hired Loudeye to perform a legally equivalent service; selling XM 120,000 compact discs on a hard drive array,²¹ XM became liable for massive copyright infringement. Neither Loudeye nor XM received permission from the songwriters to make these copies, according to the general counsel of Harry Fox, and XM appears to have indemnified Loudeye for liability for copyright infringement²². Indeed, Harry Fox appears not to have been aware of this practice until alerted by MIT. In any case, Loudeye and Harry Fox are now negotiating a license that will likely be approximately 8 cents per song. When that license agreement is concluded, which we expect to happen by July 2003, we will be able to purchase compact discs (at Loudeye's price of \$8 per CD) in MP3 format from Loudeye and launch LAMP.

¹⁸They have automatic authorization from Universal, EMI and Warner; we will need supplementary authorization from Sony and BMG.

¹⁹Rodgers & Hammerstein Org. v. UMG Recordings Inc., No. 00 Civ. 9322, 2001 U.S. Dist. LEXIS 16111, 60 U.S.P.Q.2d (BNA) 1354 (S.D.N.Y. Sept. 25, 2001).

²⁰Remember that ASCAP, BMI, and SESAC represent songwriters and publishers for purposes of *performance rights*; the Harry Fox Agency deals with *reproductions*, typically under the 17 U.S.C. § 155 (2000) "compulsory license" not at issue here.

²¹Craig Johnston, *XM Radio's Music Is Massive*, RADIO WORLD NEWSPAPER, http://www.rwonline.com/reference-room/trans-2-digital/05_rwf_xm_1.shtml (accessed May 26, 2003) (on file with author)

²²Redacted Loudeye-XM Aug. 25, 2000 "Encoding Services and Compact Disc Purchase Agreement" § 8.2, available at <http://contracts.corporate.findlaw.com/agreements/loudeye/xmsat.encode.2000.08.25.html> (accessed May 26, 2003) (on file with author)

5 Further Work

5.1 Privacy

There are substantial questions as to the appropriate amount of privacy when using LAMP, especially because anyone may listen to anyone's channel and, currently, anyone may see the names of who is controlling each channel. The following is an e-mail received by the author after playing Dvorak's Seventh Symphony on the LAMP prototype:

Dear Keith,

I just happened to be up at three in the morning because I'm pathetic and don't have a life, and I was flipping through the channels and saw that you had selected a beautiful symphony piece. Obviously, you're a man of great taste and intelligence. I'd like to get to know you better. E-mail me if you're interested.

Sex depraved freshman.

P.S. Could you send me a photo of yourself?

Clearly, many users would be uncomfortable receiving e-mails like the above as a result of playing music on LAMP. Is seeing the names of other users like being able to get a record of others' library use? Or is it like visiting a library and seeing who is there? There are significant benefits to community by seeing others' music preferences, but even so we plan an anonymity option before launching the service for real.

5.2 Replicating LAMP

I hope LAMP will be replicable at other universities and in some municipalities, but it remains to be seen whether this will be possible. If the performing rights organizations are unhappy with our use of licenses for an on-demand service, they may be less willing to grant cable television performance licenses to other universities or municipalities. Additionally, our purchase agreement involving Harry Fox appears to be the first of its kind, ever, even though other organizations (such as XM) have made purchases that appear to have required such an agreement. But if Harry Fox and Loudeye sign a general license, there should be no problem with other universities or municipalities also making purchases from Loudeye.

A more serious question concerns whether other universities or municipalities have the resources (and control of their own cable systems) to be able to devote cable channels to a project like this, and whether users at MIT will actually want to listen to music over their televisions. We will have to see.

6 Acknowledgements

LAMP is deeply indebted to the financial support of Microsoft Research via the MIT-Microsoft iCampus partnership, which has funded LAMP in two rounds since February 2002. Rebecca Bisbee and Jessica Strong from iCampus have been tireless supporters. Randy Winchester and Jon Ward of MIT Cable have provided us with much technical assistance and hotly-contested resources, motivated only by the goal of advancing a collegiate environment with experimental services available to MIT students, and I am deeply grateful to attend a university so supportive (financially and morally) of crazy projects as MIT. Caroline Niziolek provided the artwork, and Ann Hammersla, MIT's senior counsel for intellectual property, and Professor William Fisher of Harvard Law School have provided invaluable assistance with the legal issues. I also wish to acknowledge the Illinois Mathematics and Science Academy for supporting the "Luxo ACA Music Project," an early ancestor of LAMP, back in 1996 and 1997.

Finally, LAMP would not be possible without the tireless work of my partner in the project, Joshua Mandel, and the string-pulling, mature supervision, and continual patronage of our advisors, David Mitchell of Microsoft and Professor Hal Abelson of MIT.

Appendix B: LAMP Proposal to Allow Creation of Indivisible Three-Song Ephemeral Recordings

Keith J. Winstein (*keithw@mit.edu*)

February 23, 2004

Introduction

Since November 2001, Joshua Mandel and I have worked with a \$60,000 MIT-Microsoft iCampus grant to build a better music library at MIT. In pursuit of this goal, MIT has acquired cable television transmission licenses to substantially all songs copyrighted in the United States, and we have built a system to allow students and faculty to reserve cable TV channels and act as disc jockeys to play recordings from the Library Access to Music Project collection. The only remaining difficulty has been in acquiring recordings that are technically feasible to broadcast. Physical compact discs themselves are very unwieldy to manage in large numbers, and so our efforts have focused on acquiring hard-drive files to store on LAMP's broadcast equipment.

In this document, I propose a new method of obtaining these files that will accomplish LAMP's goals in a practical manner, satisfy the law, and present acceptable risk exposure to the Institute. In particular, I propose the following solution: LAMP will purchase physical CDs. Using the Copyright Act's exemption for "ephemeral recordings,"¹ we will make copies of songs onto the hard drive of LAMP's broadcast equipment. Each ephemeral recording will be a single MP3 file embodying a "transmission program" consisting of *three tracks* from a particular CD.

LAMP's users, who act as the system's disc jockeys, will only be able to play *entire* transmission programs — that is, entire three-song chunks — "in sequence and as a unit," the language of the law's definition of "transmission program."² No fast-forwarding, rewinding, or skipping will be permitted. After six months, we will repeat the process of copying three-song chunks, with a new and distinct set of "transmission programs."

I additionally propose that we ask the Harry Fox Agency for permission to reproduce its publishers' musical works in "server copies." If they grant permission, we will be able to make use of the more generous ephemeral recording exemption for educational users of sound recordings only,³ meaning we will be able to keep our ephemeral recordings for seven years, instead of just six months. (Even if the Harry Fox Agency does not grant permission, we will be able to use the more generous seven-year provision in section 112(b) on classical music where the underlying composition is uncopyrighted.)

LAMP's previous attempts to acquire recordings

From September 2002 through October 2003, MIT's attorneys and I negotiated to buy MP3s from the Loudeye Corporation, which represented and warranted that it was licensed to sell them. We eventually purchased \$30,000 worth of MP3s from the company. Unfortunately, on the day LAMP opened and began letting students and faculty broadcast music over MIT cable, Loudeye's general counsel called to say the company had made a huge mistake — contrary to its contractual representations, it did not have permission to make the copies it had sold us. As a result, we closed

¹17 U.S.C. § 112(a)(1) (2000).

²17 U.S.C. § 101 (2000).

³17 U.S.C. § 112(b) (2000).

LAMP in November 2003 until we could find another economical means of acquiring music.

Since early November, we have attempted to negotiate temporary licenses with the five major record labels for them either to bless our Loudeye copies or to provide us with hard-drive recordings of their own. So far, these negotiations have not been successful. Even if we did successfully receive hard-drive recordings from one record label in order to launch a pilot entertainment service, this would not satisfy our goal of providing a broad library of music much more than the MIT libraries would be satisfied if their collection could only include books from Random House.

Comparison with academic Napster service

When we started the MIT LAMP project in 2001, record labels had no licensing programs or services for interactive digital transmissions to the public. But since 2001, several services have arisen with licenses from the record labels, among them Roxio's Napster 2.0 and Apple's iTunes Music Store.

Why not, then, abandon MIT cable television, and reconfigure LAMP under one of these newly-offered digital transmission licensing regimes? There are several reasons why the new digital transmission licenses and services, while more permissive than what was previously available, are still unhelpful for our goal: to open an academic music library 24 hours a day, accessible across the MIT campus in dormitory rooms, lecture halls, and faculty offices.

Too expensive. Pennsylvania State University and the University of Rochester have negotiated agreements with Napster 2.0 to provide their students access to streaming songs through the service.⁴ I understand from discussions with the record labels that the price paid by the universities for these services is about \$2 per student per month. If this service were offered to MIT undergraduates for nine months out of the year, it would cost the Institute about \$75,000 per year. If offered to all MIT students for nine months a year, it would cost about \$200,000 a year. By comparison, the total "cable television" fee MIT pays to ASCAP, BMI, and SESAC is less than 15 cents per dormitory resident *per year*, for a total of less than \$600 a year.

Collection too limited. The collection of recordings that the record labels and recording artists have made available to Napster and iTunes is not suitable for an academic library. The available recordings are heavily biased toward recent, heavily promoted releases. Classical and jazz recordings, or even pop recordings made before the 1980s, are scarce. The collection available to LAMP by purchasing physical CDs at a record store (also, the collection archived by Loudeye) is much, much larger, especially in this area.

Technology is a close call. It is not clear that the Internet streaming services are in fact more convenient than closed-circuit television transmission. The streaming services, such as Napster 2.0, are only available on Microsoft Windows. LAMP, by contrast, is only available on devices that can receive television audio (principally TVs, VCRs, and stereo receivers).

A search of the "mit.edu" name service zone on Feb. 18, 2004 revealed 11,000 computers registered as running Microsoft Windows, 4,000 computers registered as Apple Macintoshes, and about 4,000 computers registered as running some form of Unix or GNU/Linux.

⁴Amy Harmon, *Penn State Will Pay To Allow Students To Download Music*, N.Y. TIMES, Nov. 7, 2003, at A1; N.Y. *College Offers Students Napster Deal*, Associated Press, Feb. 5, 2004, LEXIS, News Wires File.

In student dormitory rooms, Microsoft Windows is probably more prevalent than television-audio receivers, but not by much. In student lounges, the opposite is true. In classrooms, televisions are far more prevalent than Microsoft Windows computers. In faculty offices, Windows is probably more common than television-audio receivers.

A receiver for LAMP's signal costs about \$60 for students who do not have one; a Microsoft Windows computer is far more expensive for students who do not have one. It's a close call which situation is preferable: requiring students without Microsoft Windows, of which there are a substantial number, who want to access LAMP's music library to find access to such a computer, or requiring students without a television-audio receiver to pay about \$60 for one.

Considering that a bulk subscription to a digital transmission service (such as Napster 2.0) would *every year* cost MIT more than the entire *total cost* of LAMP, would be accessible only on computers with Microsoft Windows, and would have a less suitable collection for an academic library, it appears to us that the LAMP model, involving analog closed-circuit cable television transmissions, is still a useful one to pursue.

The ephemeral recording exception in section 112

Our goal is to make 3,000 CDs available to MIT students and faculty in their rooms and offices. Unfortunately, technical constraints generally require that for a library of this size to be practical, someone must copy the contents of physical CDs into computer files. The copying party can be the record label itself, us, or a third party (e.g., Loudeye), but without hard-drive copies, the large robot arms necessary to host a 3,000-CD library are impractical, and cost hundreds of thousands of dollars upfront and more to maintain.

In the radio industry, it is very common to manage a large library of CDs by copying them all onto hard drives.⁵ A legal difficulty arises with these copies, because transmitting organizations licensed to *broadcast* a song are rarely licensed to *reproduce* the song. To deal with this problem, in 1976 Congress granted transmitting organizations an "ephemeral recording" privilege:

Notwithstanding the provisions of section 106, and except in the case of a motion picture or other audiovisual work, it is not an infringement of copyright for a transmitting organization entitled to transmit to the public a performance or display of a work, under a license, including a statutory license under section 114(f), or transfer of the copyright or under the limitations on exclusive rights in sound recordings specified by section 114(a), or for a transmitting organization that is a broadcast radio or television station licensed as such by the Federal Communications Commission and that makes a broadcast transmission of a performance of a sound recording in a digital format on a nonsubscription basis, to make no more than one copy or phonorecord of a particular transmission program embodying the performance or display, if —

⁵Engineers I talked to at WEZS, WMJX, WROR, WKLD, WBOS, WDKK, and WODS all told me that "digital storage systems" are extremely common in the radio industry, and many stations now routinely copy all CDs onto a commercial Audiovault, Mediatouch, or Prophet hard-drive system before broadcasting.

(A) the copy or phonorecord is retained and used solely by the transmitting organization that made it, and no further copies or phonorecords are reproduced from it; and

(B) the copy or phonorecord is used solely for the transmitting organization's own transmissions within its local service area, or for purposes of archival preservation or security; and

(C) unless preserved exclusively for archival purposes, the copy or phonorecord is destroyed within six months from the date the transmission program was first transmitted to the public.⁶

A "transmission program," in turn, is defined as "a body of material that, as an aggregate, has been produced for the sole purpose of transmission to the public in sequence and as a unit."⁷ Note also that "The term 'copies' includes the material object, other than a phonorecord, in which the work is first fixed."⁸ That is, "to make no more than one copy or phonorecord of a particular transmission program," means to assemble the one and only copy of the transmission program from its source material.

As the legislative history explains:

This is the problem of what are commonly called "ephemeral recordings": copies or phonorecords of a work made for purposes of later transmission by a broadcasting organization legally entitled to transmit the work. In other words, where a broadcaster has the privilege of performing or displaying a work either because he is licensed or because the performance or display is exempted under the statute, the question is whether he should be given the additional privilege of recording the performance or display to facilitate its transmission... Subsection (a) permits the transmitting organization to make "no more than one copy or phonorecord of a particular transmission program embodying the performance or display." A "transmission program" is defined in section 101 as a body of material produced for the sole purpose of transmission as a unit. Thus, under section 112(a), a transmitter could make only one copy or phonorecord of a particular "transmission program" containing a copyrighted work, but would not be limited as to the number of times the work itself could be duplicated as part of other "transmission programs."⁹

How is this proposal different?

MIT's attorneys have previously decided, generally, that it would be too risky for LAMP to rest on the proposition that the ephemeral recording exceptions permit LAMP to make hard-drive copies of each song individually on a CD, allowing disc jockeys to select individual songs and fast-forward, pause, and rewind them.

⁶17 U.S.C. § 112(a)(1) (2000).

⁷17 U.S.C. § 101 (2000).

⁸*Id.*

⁹H.R. REP. NO. 94-1476 (reprinted in 17 U.S.C. app. (2000)).

In particular, they believe it is not well-settled whether copying a single song (that is, a single copyrighted work) from a CD to a hard drive can constitute making a copy of a “transmission program” — that is, “a *body* of material that, as an *aggregate*, has been produced for the sole purpose of transmission to the public *in sequence* and *as a unit*.”¹⁰

The critical modifications in the instant proposal are: (a) the transmission programs that we record onto hard disks will consist of three songs each, making them a “body of material,” an “aggregate,” and a “sequence,” and (b) disc jockeys will not be allowed to fast-forward, rewind, or skip within transmission programs, enforcing a restriction that they be transmitted through MIT’s cable TV system “in sequence and as a unit.” These changes make our proposed activities much closer to the examples contemplated when section 112 was enacted.

Are three songs, indivisible, a transmission program?

In preparing this proposal, I reviewed every case published in West’s *Federal Supplement* and *Federal Reporter*, and the *United States Reports*, as well as every *Federal Register* since 1980 and *Congressional Record* since 1985, every section in the *United States Code Service* and *Code of Federal Regulations*, and every law review article indexed by Lexis-Nexis, that mentioned transmission programs, ephemeral recordings, or ephemeral copies. I also consulted registration records of the Copyright Office.

To summarize, there is no slam-dunk answer to the question, “Are three songs, indivisible, a transmission program?” Nonetheless, the statutory requirements for a “transmission program” have been interpreted broadly, and our proposed indivisible three-song transmission programs reflect an abundance of caution in bending over backwards to be assured of satisfying the text of the statute. Finally, the balance of equities tilts in favor of including our proposed indivisible three-song programs within the definition of “transmission programs.”

The statutory requirements for a “transmission program” have been interpreted broadly. In an ephemeral recording context, the Second Circuit has construed “transmission program” to include a single 30-minute episode of the television show *Hard Copy*. “Because the TV stations had the right to broadcast Agee’s reproduced or altered sound recording ‘under the limitations on exclusive rights in sound recordings specified by section 114(a),’ the stations are protected by the ephemeral recording exemption.”¹¹

Cases about what constitutes a “transmission program” also arise in the context of copyright registration and deposit, because the Copyright Office’s regulations exempt “Motion pictures that consist of television transmission programs” from the deposit requirements.¹² In this context, a federal district court construed “transmission program” to apply to a single 90-minute television news broadcast produced by a television station:

Congress has defined “transmission program” as “a body of material that, as an aggregate, has been produced for the sole purpose of transmission to the public in sequence

¹⁰17 U.S.C. § 101 (2000) (emphasis added).

¹¹Agee v. Paramount Communications, Inc., 59 F.3d 317, 326–7 (2d Cir. 1995).

¹²37 C.F.R. § 202.19(c)(12) (2003).

and as a unit.” The legislative history shows that this definition encompasses non-syndicated radio and television programs. Live television news broadcasts clearly fall into the category of unpublished transmission programs.¹³

Additionally, the Copyright Office’s regulations discuss “transmission programs” as including “a regularly scheduled newscast or on-the-spot coverage of news events.”¹⁴

I should note that the two most famous cases to discuss “server copies” of recorded music — the closest technical analog to what we propose here — did not deal with transmission programs or with the ephemeral recording exemption, because they involved interactive digital transmission situations where the exemption was inapplicable on the text of the statute.¹⁵

In contrast to the facts of *MP3.com* and *Rodgers & Hammerstein*, we are dealing with analog transmissions initiated to listeners of MIT’s closed-circuit cable television system by a limited number of disc jockeys (who, in our service, are students using Web browsers). The ephemeral recording privilege that was inapplicable to MP3.com’s and UMG Recordings’ “server copies,” because those entities made interactive digital transmissions that implicated section 114, is applicable to our analog transmissions.

The proposed indivisible three-song transmission programs reflect an abundance of caution in satisfying the statute’s requirements. The privilege in section 112(a) is “to make no more than one copy or phonorecord of a particular transmission program embodying the performance.” We propose to make exactly one copy of each indivisible three-song program that will be available for disc jockeys to broadcast. After six months, we will delete the copy and, out of an abundance of caution, *never make it again.* We will not “re-rip” the transmission program after deleting it. Rather, we will make one copy each of a different set of transmission programs, each including a different sequence of songs.

We also must contend with the definitional language of section 101, which gives “transmission program” as “a body of material that, as an aggregate, has been produced for the sole purpose of transmission to the public in sequence and as a unit.” Unlike our previous proposal, to create single-song “transmission programs” that would be divisible (e.g., disc jockeys would have been able to fast-forward, rewind, and skip), this proposal is much more cautious.

In particular, we satisfy explicitly all of the statutory constraints:

“body of material”? Yes. Each transmission program will contain three distinct works, i.e., musical works and sound recordings.

“aggregate”? Yes. The songs of the program will be inseparable.

¹³*Pac. and S. Co. v. Duncan*, 572 F. Supp. 1186, 1197–8 (N.D. Ga. 1983), *aff’d in part, rev’d in part*, 744 F.2d 1490 (11th Cir. 1984) (citations omitted).

¹⁴37 C.F.R. § 202.22(e)(2) (2003).

¹⁵*UMG Recordings, Inc. v. MP3.com, Inc.*, 92 F. Supp. 2d 349 (S.D.N.Y. 2001); *Rodgers & Hammerstein Org. v. UMG Recordings Inc.*, No. 00 Civ. 9322, 2001 U.S. Dist. LEXIS 16111, 60 U.S.P.Q.2d (BNA) 1354 (S.D.N.Y. Sept. 25, 2001). Two subsequent decisions in the *MP3.com* case describe the prohibited reproductions explicitly as “server copies.” See *Copyright.net Music Publ’g LLC v. MP3.com*, 256 F. Supp. 2d 214 (S.D.N.Y. 2003); *Country Rd. Music, Inc. v. MP3.com, Inc.*, 279 F. Supp. 2d 325 (S.D.N.Y. 2003).

“produced for the sole purpose of transmission to the public in sequence and as a unit”? Yes. The system’s disc jockeys will not be able to play individual songs from within a particular transmission program, re-order the songs within a transmission program, fast-forward or rewind sections of a transmission program, or skip about within a transmission program. They will only be able to commence a three-song program, and must wait for it to play in its entirety before another begins.

Additionally, our usage is similar to what we envision to have been the primary legislative intent in drafting section 112: radio stations who wished to pre-record programs (several songs in a row, sometimes with spoken announcements in-between) for disc jockeys to more easily broadcast to listeners. What we propose is similar. LAMP’s disc jockeys could walk to the library’s physical location, physically choose from the several thousand CDs we plan to buy, and load them into physical CD drives. To make it easier, we propose to pre-record indivisible three-song “transmission programs” and allow the disc jockeys to queue the programs up for broadcasting from consoles on their own computers. To listeners, the experience is identical — except that, unlike with physical CDs present, listeners will only hear indivisible three-song programs and will not be able to successfully request that disc jockeys fast-forward, rewind, or skip immediately to another song.

The balance of equities tips in our favor. MIT has purchased the rights to broadcast the songs we want to broadcast. LAMP is willing to pay anybody who can legitimately sell us recordings in hard-drive (e.g., MP3) format, so that we do not need to make ephemeral recordings in order to broadcast the songs.

With this proposition to buy hard-drive-format music for analog broadcasts, we have approached Loudeye Corp. (who represented they did have this permission, sold us \$30,000 of recordings, and then disclosed they did not have permission to sell them to us), Apple’s iTunes Music Store, Roxio’s Napster 2.0, FullAudio’s MusicNow, and the five major record labels. After three months, we have not been able to make a successful deal to purchase hard-drive recordings from anybody.

If we pursue this proposal to construct indivisible three-song ephemeral recordings, in the end we will own and have paid for 3,000 CDs. We will have purchased licenses from the musical work copyright owners to broadcast those CDs. And “the practical exigencies of broadcasting,”¹⁶ (as evidenced by the near-universal practice of commercial radio stations) dictate that we somehow turn those physical CDs into ephemeral recordings: “copies or phonorecords of a work made for purposes of later transmission by a broadcasting organization legally entitled to transmit the work.”¹⁷

Requesting permission from the Harry Fox Agency

The Harry Fox Agency, the licensing arm of the National Music Publishers Association, has indicated a willingness to consider a special request for permission for MIT to make server copies of musical works it licenses. For works where the musical work is not copyrighted (e.g., most classical music) or where we have permission to make a “server copy,” LAMP can benefit from

¹⁶H.R. REP. NO. 94-1476.

¹⁷*Id.*

the seven-year ephemeral-recording privilege in section 112(b), instead of the six-month privilege of section 112(a).

Section 112(b) applies to “nonprofit organization entitled to transmit a performance or display of a work, under section 110(2) or under the limitations on exclusive rights in sound recordings specified by section 114(a).” Because we do not appear to qualify for section 110(2) (a distance-learning exemption), section 112(b) therefore only applies to us insofar as we are entitled to transmit performances “under the limitations on exclusive rights in sound recordings specified by section 114(a),” that is, the lack of an analog-performance right for sound recordings.

I recommend that we pursue a proposal with Harry Fox to allow us to make a single server copy of works they license. The Harry Fox general counsel, Jacqueline Charlesworth, has indicated to me that approval from almost all music publishers is likely, and the price will probably be the statutory “mechanical license” rate (in section 115), now 8.5 cents per song.

Of the 3,000 CDs that we plan to purchase, probably more than 2,000 will include some copyrighted musical works. Estimating 15 tracks per CD, times 2,000 CDs, times 8.5 cents per track, works out to a total price of \$2,550. This is well within our budget and, in return, we will be able to retain transmission programs including only licensed musical works for seven years instead of only six months.

Conclusion

LAMP aims to build a better library of music at MIT. Unlike traditional libraries, we want LAMP to be open 24 hours a day, and we want its collection to be accessible at many physical locations across campus. To accomplish our goals, we must be able to acquire a broad collection able to be practically searched and broadcast across campus. We recognize that we are in uncharted territory, and from the start our goal has been absolute and assured compliance with the law:

I want to stress, then, that I am only interested in doing this proposal if MIT is confident it would not end in a huge lawsuit. As I see it, this is an opportunity for good PR from all sides, including the media industries, to show the positive effects of collaboration and applying technology to the educational process and media distribution, and this would be the only spirit under which I think it is advisable to pursue this.¹⁸

Based on our experiences so far, the only method that appears both clearly legal and practical is the purchase of physical CDs and the construction of ephemeral recordings. By making each “transmission program” an indivisible sequence of three songs, we will reflect an abundance of caution to be assured of compliance with the statutory requirements and equities. Coupled with a proposal to the Harry Fox Agency, this method will result in the creation of a never-closing, always-close-by music library at MIT.

¹⁸Keith Winstein, *Online Music & Movie Library for MIT*, Nov. 12, 2001, at <http://web.mit.edu/keithw/music> (original iCampus proposal for LAMP; approved in 2001 funding cycle, renewed in 2002 funding cycle).

Appendix C: Source Code Listings

LAMP Display

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <sched.h>
#include <math.h>
#include <malloc.h>

#include <sys/io.h>

#include <sys/ioctl.h>
#include <linux/rtc.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/un.h>

#include <sys/poll.h>

#include <sys/mman.h>

extern "C" {
#include <ppm.h>
}

#include "display.h"

const int WIDTH = 640;
const int HEIGHT = 480;

void draw_demotext( LampDisplay *mydisplay );
void recode( char *string );

pixel **lampback_array;
FILE *lampback;
int image_rows, image_cols;
pixval maxval;

void reread( void )
{
    if ( (lampback = fopen( "lampback.ppm", "r" )) == NULL ) {
        perror( "fopen" );
        exit( 1 );
    }

    lampback_array = ppm_readppm( lampback, &image_cols, &image_rows, &maxval );

    fclose( lampback );

    if ( (image_cols != WIDTH) || (image_rows != HEIGHT) ) {
        fprintf( stderr, "Bad image size.\n" );
        exit( 1 );
    }
}

int main( int argc, char *argv[] )
{
```

```

LampDisplay *mydisplay;

ppm_init( &argc, argv );

if ( (lampback_array = ppm_allocarray( WIDTH, HEIGHT )) == NULL ) {
    perror( "ppm_allocarray" );
    exit( 1 );
}

reread();

mydisplay = new LampDisplay( WIDTH, HEIGHT );
/* check depth */
printf( "Bits per pixel: %i\n", mydisplay->bits_per_pixel );
if ( mydisplay->bits_per_pixel != 16 ) {
    fprintf( stderr, "Sorry, we don't support that bit depth.\n" );
    exit( 1 );
}

mydisplay->ppm_import( lampback_array, 1.0 );
mydisplay->update();

draw_demotext( mydisplay );

ppm_freearray( lampback_array, HEIGHT );

if ( fclose( lampback ) ) {
    perror( "fclose" );
    exit( 1 );
}
}

void draw_demotext( LampDisplay *mydisplay )
{
    Font newfont, bigfont;
    XFontStruct *newfontstruct, *bigfontstruct;
    int rtc;
    struct timeval thetime, lasttime;
    struct timezone useless;

    struct sched_param myparams;

    int realtime = atoi( getenv( "REALTIME" ) );

    int mysocket = socket( PF_UNIX, SOCK_DGRAM, 0 );

    struct pollfd ufds[ 2 ];

    struct sockaddr_un mysockaddr;

    mysockaddr.sun_family = AF_UNIX;
    strcpy( mysockaddr.sun_path, "/tmp/lamp-display-socket" );

    unlink( "/tmp/lamp-display-socket" );

    umask( 0 );

    if ( bind( mysocket, (struct sockaddr *) (&mysockaddr), sizeof( mysockaddr ) ) != 0 ) {
        perror( "bind" );
        exit( 1 );
    }

    if ( realtime ) {
        myparams.sched_priority = sched_get_priority_max( SCHED_FIFO );
        if ( sched_setscheduler( 0, SCHED_FIFO, &myparams ) == -1 ) {

```

```

    perror( "setscheduler" );
    exit( 1 );
}

if ( mlockall( MCL_CURRENT | MCL_FUTURE ) != 0 ) {
    perror( "mlockall" );
    exit( 1 );
}

rtc = open( "/dev/rtc", O_RDONLY );
if (rtc == -1) {
    perror("/dev/rtc");
    exit(errno);
}

ioctl(rtc, RTC_IRQP_SET, 8192);
ioctl(rtc, RTC_PIE_ON, 0);

if ( iopl( 3 ) != 0 ) {
    perror( "iopl" );
    exit( 1 );
}
}

char skyline[1024] = "";

char chan_strings[16][1024];
char save_strings[16][1024];
char username[16][1024];
char chan_info[16][1024];
char save_username[16][1024];
char timing[16][1024];
char save_timing[16][1024];
float opacity[16];
int fade[16];
time_t timeout[16];

for ( int j = 0; j < 16; j++ ) {
    opacity[ j ] = 1.0;
    fade[ j ] = 0;
    timeout[ j ] = 0;
    strncpy( chan_strings[ j ], "", 1024 );
    strncpy( save_username[ j ], "", 1024 );
    strncpy( chan_info[ j ], "", 1024 );
    strncpy( username[ j ], "", 1024 );
    strncpy( save_username[ j ], "", 1024 );
    strncpy( timing[ j ], "", 1024 );
    strncpy( save_timing[ j ], "", 1024 );
}

int string_lengths[16];
int save_lengths[16];
int info_lengths[16];
int timing_lengths[16];
int save_timing_lengths[16];

int margin = 85;

char demostring[200];

Pixmap mypixmap, virgin_screen, demo2pixmap, myrealpixmap;
int demowidth;
int i = 0;

```

```

XGCValues myvalues;
myvalues.background = ((205 << 8) & 63488) | ((219 << 3) & 2016)
    | (240 >> 3);
XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCBackground, &myvalues );

newfont = XLoadFont( mydisplay->mydisplay,
    "-bluesky-cmss17-medium-r-normal--0-200-0-0-p-0-adobe-fontspecific" );
newfontstruct = XQueryFont( mydisplay->mydisplay, newfont );

bigfont = XLoadFont( mydisplay->mydisplay,
    "-bluesky-cmss17-medium-r-normal--0-400-0-0-p-0-adobe-fontspecific" );
bigfontstruct = XQueryFont( mydisplay->mydisplay, bigfont );

int skylinewidth = 0;
float skylineopacity = 1.0;
char skylinefade = 0;

XSetFont( mydisplay->mydisplay, mydisplay->mygc, newfont );

snprintf( demostrating, 200, "63:%c", 160 );

demowidth = XTextWidth( newfontstruct, demostrating, strlen( demostrating ) );

mypixmap = XCreatePixmap( mydisplay->mydisplay, mydisplay->mywindow, mydisplay->width,
    subheight, 16 );

myrealpixmap = XCreatePixmap( mydisplay->mydisplay, mydisplay->mywindow, mydisplay->width,
    subheight, 16 );

demo2pixmap = XCreatePixmap( mydisplay->mydisplay, mydisplay->mywindow,
    mydisplay->width - demowidth - margin, subheight, 16 );

virgin_screen = XCreatePixmap( mydisplay->mydisplay, mydisplay->mywindow, mydisplay->width,
    subheight, 16 );

myvalues.foreground = myvalues.background;
XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );
XFillRectangle( mydisplay->mydisplay, mydisplay->mywindow, mydisplay->mygc, 0, mydisplay->height - subheight,
    mydisplay->width, subheight );

XCopyArea( mydisplay->mydisplay, mydisplay->mywindow, virgin_screen, mydisplay->mygc, 0,
    mydisplay->height - subheight, mydisplay->width, subheight, 0, 0 );
XCopyArea( mydisplay->mydisplay, virgin_screen, myrealpixmap, mydisplay->mygc, 0, 0,
    mydisplay->width, subheight, 0, 0 );
XCopyArea( mydisplay->mydisplay, virgin_screen, mypixmap, mydisplay->mygc, 0, 0,
    mydisplay->width, subheight, 0, 0 );
XCopyArea( mydisplay->mydisplay, virgin_screen, demo2pixmap, mydisplay->mygc, 0, 0,
    mydisplay->width - demowidth - margin, subheight, 0, 0 );

ufds[ 0 ].fd = mysocket;
if ( realtime ) {
    ufds[ 1 ].fd = rtc;
} else {
    ufds[ 1 ].fd = STDIN_FILENO;
}

for ( int k = 0; k < 16; k++ ) {
    string_lengths[ k ] = XTextWidth( newfontstruct, chan_strings[ k ], strlen( chan_strings[ k ] ) );
    save_lengths[ k ] = XTextWidth( newfontstruct, save_strings[ k ], strlen( save_strings[ k ] ) );
    info_lengths[ k ] = XTextWidth( newfontstruct, chan_info[ k ], strlen( chan_info[ k ] ) );
    timing_lengths[ k ] = XTextWidth( newfontstruct, timing[ k ], strlen( timing[ k ] ) );
    save_timing_lengths[ k ] = XTextWidth( newfontstruct, save_timing[ k ], strlen( save_timing[ k ] ) );
}

myvalues.foreground = 0;

```

```

XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );

for ( int j = 63; j <= 77; j++ ) {
    snprintf( demostring, 200, "%d:%c", j, 160 );
    XDrawImageString( mydisplay->mydisplay, mypixmap, mydisplay->mygc,
        50, 18 * ( j - 62), demostring, strlen( demostring ) );
}

gettimeofday( &lasttime, &useless );

while( 1 ) {
    time_t the_time = time( NULL );
    i--;

    myvalues.foreground = myvalues.background;
    XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );
    XFillRectangle( mydisplay->mydisplay, demo2pixmap, mydisplay->mygc, 0, 0,
        mydisplay->width - demowidth - margin, subheight );

    if ( skylinefade == -1 ) {
        if ( skylineopacity > 0.0 ) {
            mydisplay->ppm_import( lampback_array, skylineopacity );
            mydisplay->update();
        } else {
            XSetFont( mydisplay->mydisplay, mydisplay->mygc, bigfont );
            float tmpopac = -skylineopacity;
            if ( tmpopac > 1.0 ) {
                tmpopac = 1.0;
            }
            myvalues.foreground = color( 0, 0, 0, tmpopac );
            XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );
            XDrawString( mydisplay->mydisplay, mydisplay->mywindow, mydisplay->mygc,
                320 - skylinewidth/2, 100,
                skyline, strlen( skyline ) );
            XSetFont( mydisplay->mydisplay, mydisplay->mygc, newfont );
        }
        skylineopacity -= 0.03;
        if ( skylineopacity < -8.0 ) {
            skylineopacity = -8.0;
            skylinefade = 1;
        }
    } else if ( skylinefade == 1 ) {
        if ( skylineopacity > 0.0 ) {
            mydisplay->ppm_import( lampback_array, skylineopacity );
            mydisplay->update();
        } else {
            XSetFont( mydisplay->mydisplay, mydisplay->mygc, bigfont );
            float tmpopac = -skylineopacity;
            if ( tmpopac > 1.0 ) {
                tmpopac = 1.0;
            }
            myvalues.foreground = color( 0, 0, 0, tmpopac );
            XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );
            XDrawString( mydisplay->mydisplay, mydisplay->mywindow, mydisplay->mygc,
                320 - skylinewidth/2, 100,
                skyline, strlen( skyline ) );
            XSetFont( mydisplay->mydisplay, mydisplay->mygc, newfont );
        }
        skylineopacity += 0.03;
        if ( skylineopacity > 1.0 ) {
            skylineopacity = 1.0;
            skylinefade = 0;
        }
    }
}

```

```

for ( int j = 0; j < 16; j++ ) {
    int length = timing_lengths[ j ] + string_lengths[ j ] + info_lengths[ j ];

    if ( fade[ j ] == 1 ) {
        opacity[ j ] += 0.03;

        if ( opacity[ j ] > 1.0 ) {
            opacity[ j ] = 1.0;
            fade[ j ] = 0;
        }
    } else if ( fade[ j ] == -1 ) {
        opacity[ j ] -= 0.03;

        if ( opacity[ j ] < 0.0 ) {
            opacity[ j ] = 0.0;
            fade[ j ] = 1;
            strcpy( chan_strings[ j ], save_strings[ j ] );
            strcpy( timing[ j ], save_timing[ j ] );
            string_lengths[ j ] = save_lengths[ j ];
            timing_lengths[ j ] = save_timing_lengths[ j ];
            strcpy( username[ j ], save_username[ j ] );
            if ( strcmp( username[ j ], "" ) == 0 ) {
                strcpy( chan_info[ j ], " " );
            } else {
                sprintf( chan_info[ j ], "(%s, 00m) ", username[ j ] );
            }
            recode( chan_info[ j ] );
            info_lengths[ j ] = XTextWidth( newfontstruct,
                                           chan_info[ j ],
                                           strlen( chan_info[ j ] ) );
        }
    }

    time_t diff = timeout[ j ] - the_time;
    if ( diff < 0 ) { diff = 0; }
    diff += 59;
    diff /= 60;

    if ( fade[ j ] != -1 ) {
        if ( strcmp( username[ j ], "" ) == 0 ) {
            strcpy( chan_info[ j ], " " );
        } else {
            sprintf( chan_info[ j ], "(%s, %dm) ", username[ j ], (int)diff );
        }
        recode( chan_info[ j ] );
    }

    if ( length > mydisplay->width - demowidth - margin ) {
        double speedbump = length / (1.5*479.52);
        myvalues.foreground = color( 160, 32, 54, opacity[ j ] );
        XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );

        XDrawString( mydisplay->mydisplay, demo2pixmap, mydisplay->mygc,
                    int(i*speedbump) % length, 18 * (j + 1),
                    timing[ j ], strlen( timing[ j ] ) );
        XDrawString( mydisplay->mydisplay, demo2pixmap, mydisplay->mygc,
                    (int(i*speedbump) % length) + length, 18 * (j + 1),
                    timing[ j ], strlen( timing[ j ] ) );

        if ( j % 2 ) {
            myvalues.foreground = color( 0, 0, 0, opacity[ j ] );
        } else {
            myvalues.foreground = color( 0x20, 0x20, 0x90, opacity[ j ] );
        }
        XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );
    }
}

```



```

XDrawString( mydisplay->mydisplay, demo2pixmap, mydisplay->mygc,
              (int(i*speedbump) % length) + timing_lengths[ j ], 18 * (j + 1),
              chan_strings[ j ], strlen( chan_strings[ j ] ) );
XDrawString( mydisplay->mydisplay, demo2pixmap, mydisplay->mygc,
              ((int(i*speedbump) % length) + length) + timing_lengths[ j ], 18 * (j + 1),
              chan_strings[ j ], strlen( chan_strings[ j ] ) );

myvalues.foreground = color( 96, 96, 96, opacity[ j ] );

XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );

XDrawString( mydisplay->mydisplay, demo2pixmap, mydisplay->mygc,
              (int(i*speedbump) % length) + timing_lengths[ j ] + string_lengths[ j ], 18 * (j + 1),
              chan_info[ j ], strlen( chan_info[ j ] ) );
XDrawString( mydisplay->mydisplay, demo2pixmap, mydisplay->mygc,
              (int(i*speedbump) % length) + length + timing_lengths[ j ] + string_lengths[ j ], 18 * (j + 1),
              chan_info[ j ], strlen( chan_info[ j ] ) );
} else {
myvalues.foreground = color( 160, 32, 54, opacity[ j ] );
XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );

XDrawString( mydisplay->mydisplay, demo2pixmap, mydisplay->mygc,
              0, 18 * (j + 1),
              timing[ j ], strlen( timing[ j ] ) );

if ( j % 2 ) {
myvalues.foreground = color( 0, 0, 0, opacity[ j ] );
} else {
myvalues.foreground = color( 0x20, 0x20, 0x90, opacity[ j ] );
}
XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );

XDrawString( mydisplay->mydisplay, demo2pixmap, mydisplay->mygc,
              timing_lengths[ j ], 18 * (j + 1),
              chan_strings[ j ], strlen( chan_strings[ j ] ) );
myvalues.foreground = color( 96, 96, 96, opacity[ j ] );
XChangeGC( mydisplay->mydisplay, mydisplay->mygc, GCForeground, &myvalues );

XDrawString( mydisplay->mydisplay, demo2pixmap, mydisplay->mygc,
              timing_lengths[ j ] + string_lengths[ j ], 18 * (j + 1),
              chan_info[ j ], strlen( chan_info[ j ] ) );
}
}

XCopyArea( mydisplay->mydisplay, demo2pixmap, mypixmap, mydisplay->mygc, 0, 0,
           mydisplay->width - demowidth - margin, subheight, 50 + demowidth, 0 );

unsigned long data;
while( 1 ) {
unsigned long difference;
char packet[ 1024 ];

ufds[ 0 ].events = POLLIN;
ufds[ 1 ].events = POLLIN;

poll( ufds, 2, 10 );

if ( ufds[ 0 ].revents & POLLIN ) {
memset( packet, 0, 1024 );
read( mysocket, packet, 1024 );

if ( packet[ 0 ] == 0 ) { /* song update */
int chan = packet[ 1 ];
if ( (chan < 63) || (chan > 78) ) {

```

```

    fprintf( stderr, "Bad chan %d\n", chan );
    exit( 1 );
}

memset( save_strings[ chan - 63 ], 0, 1024 );
strncpy( save_strings[ chan - 63 ], packet + 2, 1022 );
recode( save_strings[ chan - 63 ] );
save_lengths[ chan - 63 ] = XTextWidth( newfontstruct,
                                       save_strings[ chan - 63 ],
                                       strlen( save_strings[ chan - 63 ] ) );

if ( strcmp( save_strings[ chan - 63 ], chan_strings[ chan - 63 ] ) != 0 ) {
    fade[ chan - 63 ] = -1;
}
} else if ( packet [ 0 ] == 1 ) { /* timing update */
int chan = packet[ 1 ];
if ( (chan < 63) || (chan > 78) ) {
    fprintf( stderr, "Bad chan %d\n", chan );
    exit( 1 );
}
if ( fade[ chan - 63 ] != -1 ) {
    memset( timing[ chan - 63 ], 0, 1024 );
    strncpy( timing[ chan - 63 ], packet + 2, 1022 );
    recode( timing[ chan - 63 ] );
    strcpy( save_timing[ chan - 63 ], timing[ chan - 63 ] );
    save_timing_lengths[ chan - 63 ] = timing_lengths[ chan - 63 ]
        = XTextWidth( newfontstruct,
                    timing[ chan - 63 ],
                    strlen( timing[ chan - 63 ] ) );
} else {
    memset( save_timing[ chan - 63 ], 0, 1024 );
    strncpy( save_timing[ chan - 63 ], packet + 2, 1022 );
    recode( save_timing[ chan - 63 ] );
    save_timing_lengths[ chan - 63 ] = XTextWidth( newfontstruct,
                                                save_timing[ chan - 63 ],
                                                strlen( save_timing[ chan - 63 ] ) );
}
} else if ( packet [ 0 ] == 2 ) { /* info update */
int chan = packet[ 1 ];
if ( (chan < 63) || (chan > 78) ) {
    fprintf( stderr, "Bad chan %d\n", chan );
    exit( 1 );
}
}

memset( save_username[ chan - 63 ], 0, 1024 );
strncpy( save_username[ chan - 63 ], packet + 2, 1022 );
recode( save_username[ chan - 63 ] );

if ( strcmp( save_username[ chan - 63 ], username[ chan - 63 ] ) != 0 ) {
    fade[ chan - 63 ] = -1;
}
} else if ( packet [ 0 ] == 3 ) { /* timeout update */
int chan = packet[ 1 ];
if ( (chan < 63) || (chan > 78) ) {
    fprintf( stderr, "Bad chan %d\n", chan );
    exit( 1 );
}
}

memcpy( &(timeout[ chan - 63 ]), &(packet[ 2 ]), sizeof( time_t ) );
} else if ( packet[ 0 ] == 4 ) { /* skyline */
memset( skyline, 0, 1024 );
strncpy( skyline, packet + 1, 1023 );
recode( skyline );
skylinewidth = XTextWidth( bigfontstruct, skyline, strlen( skyline ) );
skylinefade = -1;
}

```

```

        skylineopacity = 1.0;
    } else if ( packet[ 0 ] == 5 ) { /* reread */
        reread();
    } else {
        fprintf( stderr, "Bad packet type %d\n", packet[ 0 ] );
        exit( 1 );
    }
}

if ( realtime ) {
    if ( ufds[ 1 ].revents & POLLIN ) {
        read( rtc, &data, sizeof( unsigned long ) );
    }
}

gettimeofday( &thetime, &useless );
difference = 1000000 * (thetime.tv_sec - lasttime.tv_sec) +
    (thetime.tv_usec - lasttime.tv_usec);

if ( difference > 16500 ) {
    break;
}

if ( realtime ) {
    while( 1 ) {
        if( (inb( 0x3da ) & 8) ) break;
    }
    while( 1 ) {
        if( !(inb( 0x3da ) & 8) ) break;
    }
}

gettimeofday( &lasttime, &useless );

XCopyArea( mydisplay->mydisplay, mypixmap, mydisplay->mywindow, mydisplay->mygc, 0, 0,
    mydisplay->width, subheight, 0, mydisplay->height - subheight );
XFlush( mydisplay->mydisplay );

if ( i % 1000 ) {
    XEvent ev;
    while(XPending(mydisplay->mydisplay)) XNextEvent(mydisplay->mydisplay, &ev);
}

int x, y, z;
mydisplay->getclick( &x, &y, &z );

XFreePixmap( mydisplay->mydisplay, mypixmap );
}

void recode( char *string )
{
    char *ptr;
    while ( (ptr = strchr( string, ' ' )) ) {
        *ptr = 160;
    }

    while ( (ptr = strchr( string, '_' )) ) {
        *ptr = 160;
    }
}

```

LAMP MP3 Decoder

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <malloc.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/soundcard.h>
#include <string.h>
#include <sys/types.h>
#include <sys/un.h>
#include <sys/poll.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <mad.h>
#include <sched.h>
#include <sys/mman.h>

#define SAMPLE unsigned short
const int BUFFSIZE = 352800;
#define NUM_CHANNELS 13

struct pollfd ufds[ NUM_CHANNELS + 1 ];
int frag;
int card_bufsize;
int dispsocket;
int format, stereo;
char zeros[ BUFFSIZE ];
char packet[ 1024 ];

static inline
signed int scale(mad_fixed_t sample)
{
    /* round */
    sample += (1L << (MAD_F_FRACBITS - 16));

    /* clip */
    if (sample >= MAD_F_ONE)
        sample = MAD_F_ONE - 1;
    else if (sample < -MAD_F_ONE)
        sample = -MAD_F_ONE;

    /* quantize */
    return sample >> (MAD_F_FRACBITS + 1 - 16);
}

class Channel {
public:
    int index;
    int audio_fd;
    char *buffer;
    int length;
    int position;
    int desired_position;
    char timing_string[ 256 ];
    char save_timing_string[ 256 ];
    char playing;
    char need_audio;
    char end_of_stream;
    char length_found;
    char present;
    int decodeindex;
    int playindex;
```

```

int file_fd;
size_t file_length;
size_t file_begin;
const unsigned char *mp3data;
char play_packet[ 1024 ];
char filename[ 1024 ];
char username[ 1024 ];
char song_title[ 1024 ];
unsigned int timeout;
int percent;

struct mad_stream Stream;
struct mad_frame Frame;
struct mad_synth Synth;
mad_timer_t Timer;

Channel( int i ) {
    index = i;
    char device_name[1024];
    int num = i < 8 ? i : i + 16;
    snprintf( device_name, 1024, "/dev/dsp%d", num );

    decodeindex = 0;
    playindex = 0;

    if ( ( audio_fd = open( device_name, O_WRONLY, 0 ) ) == -1 ) {
        perror( device_name );
        exit( 1 );
    }

    if ( ioctl( audio_fd, SNDCTL_DSP_SETFMT, &format ) != 0 ) {
        perror( "SNDCTL_DSP_SETFMT" );
        exit( 1 );
    }

    if ( ioctl( audio_fd, SNDCTL_DSP_STEREO, &stereo ) != 0 ) {
        perror( "SNDCTL_DSP_STEREO" );
        exit( 1 );
    }

    if ( fcntl( audio_fd, F_SETFL, O_NONBLOCK ) != 0 ) {
        perror( "fcntl" );
        exit( 1 );
    }

    ufds[ i ].fd = audio_fd;
    ufds[ i ].events = POLLOUT;

    /* set up buffer */
    buffer = (char *) malloc( BUFFSIZE );
    if ( buffer == NULL ) {
        perror( "malloc" );
        exit( 1 );
    }

    memset( buffer, 0, BUFFSIZE );

    playing = 0;
    present = 0;

    strcpy( save_timing_string, "" );
}

void play( void ) {
    memcpy( play_packet, packet, 1024 );
}

```

```

if ( (present == 0) || (playing == 0) || (end_of_stream == 0) ) {
    memset( buffer, 0, BUFFSIZE );
    decodeindex = 0;
    playindex = 0;
}

stop();

char *stringp = play_packet + 12;

char delim[2];
delim[0] = '-';
delim[1] = '\0';

strcpy( filename, strsep( &stringp, delim ) );
strcpy( username, strsep( &stringp, delim ) );
strcpy( song_title, strsep( &stringp, delim ) );

file_fd = open( filename, O_RDONLY );
if ( file_fd == -1 ) {
    fprintf( stderr, "Can't open %s\n", filename );
    perror( "open" );
    return;
}

struct stat my_stat;
if ( stat( filename, &my_stat ) < 0 ) {
    fprintf( stderr, "stat(%s) gives error.\n", filename );
    perror( "stat" );
    exit( 1 );
}

file_length = my_stat.st_size;

mp3data = (const unsigned char *) mmap( NULL, file_length, PROT_READ, MAP_SHARED, file_fd, 0 );
if ( mp3data < 0 ) {
    perror( "mmap" );
    exit( 1 );
}

// search for frame sync. assume byte-aligned
for ( unsigned int i = 0; i < file_length; i++ ) {
    if ( mp3data[ i ] == 0xff ) {
        file_begin = i;
        break;
    }
}

length_found = 0;

mad_stream_init(&Stream);
mad_timer_reset(&Timer);

mad_stream_buffer( &Stream, mp3data + file_begin,
                  file_length - file_begin );

present = 1;
need_audio = 1;
}

void length_tick ( void ) {
    if ( length_found ) {
        fprintf( stderr, "length_tick called but length already found.\n" );
        exit( 1 );
    }
}

```

```

}

int tries = 0;
while ( tries < 200 ) {
    struct mad_header Header;
    mad_header_decode(&Header, &Stream);
    mad_timer_add( &Timer, Header.duration );

    tries++;
    if ( Stream.error == MAD_ERROR_BUFLLEN ) {
        break;
    }
}

if ( Stream.error == MAD_ERROR_BUFLLEN ) {
    length_found = 1;

    length = Timer.seconds * 44100;

    int *packet_i = (int *)play_packet;

    char sendpacket[ 1024 ];

    timeout = packet_i[ 1 ];
    percent = packet_i[ 2 ];

    mad_stream_finish(&Stream);
    mad_stream_init(&Stream);
    mad_frame_init(&Frame);
    mad_synth_init(&Synth);
    mad_timer_reset(&Timer);

    mad_stream_buffer( &Stream, mp3data + file_begin,
                      file_length - file_begin );

    desired_position = (length / 100) * percent;

    int play_margin = decodeindex - playindex;
    while ( play_margin < 0 ) {
        play_margin += BUFFSIZE;
    }

    position = desired_position - (play_margin/2);

    int chan = index + 63;

    memset( sendpacket, 0, 1024 );
    sendpacket[ 0 ] = 0;
    sendpacket[ 1 ] = chan;
    strcat( song_title, " " );
    memcpy( &(sendpacket[ 2 ]), &song_title, strlen( song_title ) );
    write( dispsocket, sendpacket, 1024 );

    memset( sendpacket, 0, 1024 );
    sendpacket[ 0 ] = 2;
    sendpacket[ 1 ] = chan;
    memcpy( &(sendpacket[ 2 ]), &username, strlen( username ) );
    write( dispsocket, sendpacket, 1024 );

    memset( sendpacket, 0, 1024 );
    sendpacket[ 0 ] = 3;
    sendpacket[ 1 ] = chan;
    memcpy( &(sendpacket[ 2 ]), &timeout, sizeof( unsigned int ) );
    write( dispsocket, sendpacket, 1024 );
}

```

```

    char save_playing = playing;
    playing = 1;
    end_of_stream = 0; /* KJW Sept. 3, 2004 */

    send_timing();
    playing = save_playing;
}
}

void stop( void ) {
    if ( present ) {
        mad_synth_finish(&Synth);
        mad_frame_finish(&Frame);
        mad_stream_finish(&Stream);
        munmap( (void*)mp3data, file_length );
        close( file_fd );
    }
    playing = 0;
    present = 0;
    end_of_stream = 1;
    /* KJW changed Sept. 3, 2004 */
}

void pause( void ) {
    if ( present ) {
        playing = 0;
    }
}

void resume( void ) {
    if ( present ) {
        playing = 1;
    }
}

void play_tick( void ) {
    audio_buf_info info;

    if ( ioctl( audio_fd, SNDCTL_DSP_GETOSPACE, &info ) < 0 ) {
        perror( "ioctl SNDCTL_DSP_GETOSPACE" );
    }

    for ( int i = 0; i < info.fragments; i++ ) {
        play_frag();
    }

    send_timing();
}

void tick( void ) {
    int decode_margin = playindex - decodeindex;
    while ( decode_margin < BUFFSIZE ) {
        decode_margin += BUFFSIZE;
    }
    if ( end_of_stream || ((decode_margin > 0) && (decode_margin < 9000)) ) {
        need_audio = 0;
        return;
    } else {
        need_audio = 1;
    }

    int tries = 0;
    while ( (Timer.seconds * 44100 < desired_position) && (tries < 200) ) {
        struct mad_header Header;
        mad_header_decode( &Header, &Stream );
    }
}

```



```

    mad_timer_add( &Timer, Header.duration );
    tries++;
}

if ( Timer.seconds * 44100 < desired_position ) {
    return;
}

mad_frame_decode( &Frame, &Stream );
mad_timer_add( &Timer, Frame.header.duration );

mad_synth_frame( &Synth, &Frame );

for ( int i = 0; i < Synth.pcm.length; i++ ) {
    int LeftSample, RightSample;

    LeftSample = scale(Synth.pcm.samples[0][i]);
    RightSample = scale(Synth.pcm.samples[1][i]);

    SAMPLE the_sample = int((LeftSample + RightSample) * 0.4);

    buffer[ decodeindex ] = the_sample & 0xff;
    buffer[ decodeindex + 1 ] = the_sample >> 8;
    decodeindex += 2;
    if ( decodeindex >= BUFFSIZE ) {
        decodeindex -= BUFFSIZE;
    }
}

if ( Stream.error == MAD_ERROR_BUFLEN ) {
    need_audio = 0;
    end_of_stream = 1;
    char tmp[ 1024 ];
    sprintf( tmp, 1024, "/usr/local/bin/lamp-endofsong %d &", index + 63 );
    system( tmp );
}

decode_margin = playindex - decodeindex;
while ( decode_margin < BUFFSIZE ) {
    decode_margin += BUFFSIZE;
}
if ( (decode_margin > 0) && (decode_margin < 9000) ) {
    need_audio = 0;
    return;
} else {
    need_audio = 1;
}
}

void play_frag( void ) {
    if ( !playing ) {
        write( audio_fd, zeros, frag );
        return;
    }
    int play_margin = decodeindex - playindex;
    while ( play_margin < 0 ) {
        play_margin += BUFFSIZE;
    }
    if ( play_margin < frag - 1 ) {
        write( audio_fd, zeros, frag );
        return;
    }
    char scratch[ BUFFSIZE ];
    for ( int i = 0; i < frag; i++ ) {
        scratch[ i ] = buffer[(playindex + i) % BUFFSIZE];
    }
}

```

```

}
playindex += frag;
while ( playindex > BUFFSIZE ) {
    playindex -= BUFFSIZE;
}
write( audio_fd, scratch, frag );
position += (frag / sizeof(SAMPLE));

int decode_margin = playindex - decodeindex;
while ( decode_margin < 0 ) {
    decode_margin += BUFFSIZE;
}
if ( (decode_margin > 0) && (decode_margin < 9000) ) {
    need_audio = 0;
    return;
} else {
    need_audio = 1;
}
}

void send_timing( void ) {
    if ( end_of_stream ) {
        return;
    }

    int my_position = position;
    if ( position < 0 ) my_position = 0;

    snprintf( timing_string, 256, "[%d:%.2d/%d:%.2d] ",
              (my_position / 44100) / 60,
              (my_position / 44100) % 60,
              (length / 44100) / 60,
              (length / 44100) % 60 );

    /*
    int play_margin = decodeindex - playindex;
    while ( play_margin < 0 ) {
        play_margin += BUFFSIZE;
    }

    int buffered_bytes;
    ioctl( audio_fd, SNDCTL_DSP_GETODELAY, &buffered_bytes );

    snprintf( timing_string, 256, "[%d/%d] ", play_margin, buffered_bytes );
    */

    if ( strcmp( timing_string, save_timing_string ) != 0 ) {
        strcpy( save_timing_string, timing_string );
        char spacket[ 1024 ];
        memset( spacket, 0, 1024 );
        spacket[ 0 ] = 1;
        spacket[ 1 ] = index + 63;
        memcpy( &(spacket[ 2 ]), &(save_timing_string),
              strlen( save_timing_string ) );
        write( dispsocket, spacket, 1024 );
    }
}
};

int main( void );
int max( int a, int b );

int max( int a, int b )
{
    return (a > b) ? a : b;
}

```

```

}

int main( void )
{
    Channel *channel[ NUM_CHANNELS ];
    int mysocket = socket( PF_UNIX, SOCK_DGRAM, 0 );
    dispsocket = socket( PF_UNIX, SOCK_DGRAM, 0 );
    struct sockaddr_un mysockaddr;

    format = AFMT_S16_LE;
    stereo = 0;

    for ( int i = 0; i < NUM_CHANNELS; i++ ) {
        channel[ i ] = new Channel( i );
    }

    /* get fragment size */
    if ( ioctl( channel[ 0 ]->audio_fd, SNDCTL_DSP_GETBLKSIZE, &frag ) != 0 ) {
        perror( "SNDCTL_DSP_GETBLKSIZE" );
        exit( 1 );
    }

    fprintf( stderr, "fragment size: %d\n", frag );

    /* set up sockets */
    mysockaddr.sun_family = AF_UNIX;
    strcpy( mysockaddr.sun_path, "/tmp/lamp-play-socket" );
    unlink( "/tmp/lamp-play-socket" );
    umask( 0 );
    if ( bind( mysocket, (struct sockaddr *) (&mysockaddr), sizeof( mysockaddr ) ) != 0 ) {
        perror( "bind" );
        exit( 1 );
    }
    strcpy( mysockaddr.sun_path, "/tmp/lamp-display-socket" );
    if ( connect( dispsocket, (struct sockaddr *) (&mysockaddr), sizeof( mysockaddr ) ) != 0 ) {
        perror( "connect" );
    }

    memset( zeros, 0, BUFFSIZE );

    ufds[ NUM_CHANNELS ].fd = mysocket;
    ufds[ NUM_CHANNELS ].events = POLLIN;

    while( 1 ) {
        poll( ufds, NUM_CHANNELS + 1, -1 );

        if ( ufds[ NUM_CHANNELS ].revents & POLLIN ) { // socket
            memset( packet, 0, 1024 );
            read( mysocket, packet, 1024 );

            if ( packet[ 0 ] == 0 ) { /* play song */
                int chan = packet[ 1 ];
                if ( (chan < 63) || (chan > 63 + NUM_CHANNELS) ) {
                    fprintf( stderr, "Bad chan %d\n", chan );
                    exit( 1 );
                }

                int index = chan - 63;
                channel[ index ]->play();
            } else if ( packet[ 0 ] == 1 ) { /* stop */
                int chan = packet[ 1 ];
                if ( (chan < 63) || (chan > 63 + NUM_CHANNELS) ) {
                    fprintf( stderr, "Bad chan %d\n", chan );
                    exit( 1 );
                }
            }
        }
    }
}

```

```

    int index = chan - 63;
    channel[ index ]->stop();
} else if ( packet[ 0 ] == 2 ) { /* pause */
    int chan = packet[ 1 ];
    if ( (chan < 63) || (chan > 63 + NUM_CHANNELS) ) {
        fprintf( stderr, "Bad chan %d\n", chan );
        exit( 1 );
    }

    int index = chan - 63;
    channel[ index ]->pause();
} else if ( packet[ 0 ] == 3 ) { /* resume */
    int chan = packet[ 1 ];
    if ( (chan < 63) || (chan > 63 + NUM_CHANNELS) ) {
        fprintf( stderr, "Bad chan %d\n", chan );
        exit( 1 );
    }

    int index = chan - 63;
    channel[ index ]->resume();
} else {
    exit( 1 );
}
}

for ( int i = 0; i < NUM_CHANNELS; i++ ) {
    if ( channel[ i ]->present && (channel[ i ]->length_found == 0) ) {
        channel[ i ]->length_tick();
    }
}

for ( int i = 0; i < NUM_CHANNELS; i++ ) {
    if ( channel[ i ]->playing && channel[ i ]->need_audio && channel[ i ]->length_found ) {
        channel[ i ]->tick();
    }
}

for ( int i = 0; i < NUM_CHANNELS; i++ ) { // audio devices
    if ( ufds[ i ].revents & POLLOUT ) {
        channel[ i ]->play_tick();
    }
}
}

return 0;
}

```

I²S PXA255 Linux Device Driver

```
/*
 * I2S Audio Input for the RBX1600
 *
 * By Josh Mandel and Keith Winstein.
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/delay.h>
#include <linux/pm.h>
#include <linux/errno.h>
#include <linux/slab.h>
#include <linux/sound.h>
#include <linux/soundcard.h>
#include <linux/kmod.h>
#include <linux/proc_fs.h>

#include <asm/semaphore.h>
#include <asm/uaccess.h>
#include <asm/hardware.h>
#include <asm/dma.h>

#include <asm/irq.h>

#include "../rbx1600/dac.h"

int read_frag;
int write_frag;
int write_ptr;
unsigned long volatile last_interrupt_jiffies;
char *audio_buffer;
char fifo_emptied;

#define FRAG_SIZE 588 /* must be divisible by 4 */
#define NUM_FRAGS 4 /* one frame total */
#define BUFFER_SIZE (FRAG_SIZE * NUM_FRAGS)

DECLARE_WAIT_QUEUE_HEAD( wait );

static void rbx1600_irq(int ch, void *dev_id, struct pt_regs *regs)
{
    u32 sasr0_val, rfifos;
    u32 sadr_val;
    int write_position;
    char new_frag = 0;

    last_interrupt_jiffies = jiffies;

    SAIMR = 0; /* disable receive-FIFO interrupts */

    /* debug */
    sasr0_val = SASR0;
    rfifos = (sasr0_val & 0xf000) >> 12;
    if ( rfifos == 0 ) {
        rfifos = 16;
    }

    if ( rfifos < 6 ) { // Leaving this code out causes interrupt lockup!
        udelay( 120 ); // 5.3 samples @ 44.1 kHz
        SAIMR = 16;
        // printk( KERN_INFO "rfifos is %d; why an interrupt?\n", rfifos );
    }
}
```

```

    return;
}

if ( fifo_emptied && (rfifos > 12) ) {
    printk( KERN_INFO "Warning: receive FIFO entries = %d/16\n", rfifos );
}

while ( 1 ) {
    sasr0_val = SASR0;
    rfifos = (sasr0_val & 0xf000) >> 12;
    if ( rfifos == 0 ) {
        rfifos = 16;
    }

    if ( rfifos == 1 ) {
        fifo_emptied = 1;
        break;
    }

    sadr_val = SADR;

    write_position = (FRAG_SIZE * write_frag) + write_ptr;

    audio_buffer[ write_position ]      = (sadr_val & 0xff000000) >> 24;
    audio_buffer[ write_position + 1 ]  = (sadr_val & 0x00ff0000) >> 16;
    audio_buffer[ write_position + 2 ]  = (sadr_val & 0x0000ff00) >> 8;
    audio_buffer[ write_position + 3 ]  = (sadr_val & 0x000000ff);

    write_ptr += 4;

    if ( write_ptr == FRAG_SIZE ) {
        write_ptr = 0;
        write_frag = (write_frag + 1) % NUM_FRAGS;
        new_frag = 1;
    }
}

SAIMR = 16; /* enable receive-FIFO interrupts */

if ( new_frag ) {
    wake_up_interruptible_sync( &wait );
}
}

int rbx1600_audio_open( struct inode *inode, struct file *file )
{
    MOD_INC_USE_COUNT;
    read_frag = 0;
    write_frag = 0;
    write_ptr = 0;
    fifo_emptied = 0;
    last_interrupt_jiffies = jiffies;

    audio_buffer = (char *) kmalloc( BUFFER_SIZE, GFP_KERNEL );
    if ( audio_buffer == NULL ) {
        printk( KERN_INFO "I2S: Couldn't allocate audio buffer.\n" );
        MOD_DEC_USE_COUNT;
        return -ENOMEM;
    }

    if ( request_irq( IRQ_I2S, rbx1600_irq, SA_INTERRUPT, "I2S", NULL ) ) {
        printk( KERN_INFO "I2S: Couldn't allocate IRQ.\n" );
        kfree( audio_buffer );
        MOD_DEC_USE_COUNT;
        return -EBUSY;
    }
}

```

```

}

return 0;
}

int rbx1600_audio_release( struct inode *inode, struct file *file )
{
    free_irq( IRQ_I2S, NULL );
    kfree( audio_buffer );
    audio_buffer = 0;
    MOD_DEC_USE_COUNT;
    return 0;
}

ssize_t rbx1600_audio_read( struct file *filp, char *buff,
                           size_t count, loff_t *offp )
{
    if ( count < FRAG_SIZE ) {
        return -EFAULT;
    }

    // wait_event_interruptible( wait, (write_frag != read_frag) );

    interruptible_sleep_on_timeout( &wait, 3 );

    if ( write_frag == read_frag ) {
        printk( KERN_INFO "write_frag == read_frag (1x)\n" );
        interruptible_sleep_on_timeout( &wait, 3 );
    }

    if ( ( write_frag == read_frag )
        || ( ( last_interrupt_jiffies - jiffies > 3 )
            && ( last_interrupt_jiffies - jiffies < 259200000 ) ) ) {
        /* a month */
        if ( write_frag == read_frag )
            printk( KERN_INFO "write_frag == read_frag (2x)\n" );

        printk( KERN_INFO "Resetting I2S device (no interrupts for %d jiffies).\n",
                last_interrupt_jiffies - jiffies );

        set_GPIO_mode(GPIO028_BITCLK_I2S_MD);
        set_GPIO_mode(GPIO029_SDATA_IN_I2S_MD);
        set_GPIO_mode(GPIO030_SDATA_OUT_I2S_MD);
        set_GPIO_mode(GPIO031_SYNC_I2S_MD);
        set_GPIO_mode(GPIO032_SDATA_IN1_AC97_MD);

        CKEN |= CKEN8_I2S;
        SACRO = SACRO_RST;
        mdelay(1);
        SACRO = SACRO_ENB | SACRO_RFTH(6) | SACRO_TFTH(6);

        SAIMR = 16; /* enable receive-FIFO interrupts */
        SACR1 = 0; /* I2S operation */

        return -EINTR;
    }

    copy_to_user( buff, audio_buffer + (read_frag * FRAG_SIZE), FRAG_SIZE );
    read_frag = (read_frag + 1) % NUM_FRAGS;

    return FRAG_SIZE;
}

static struct file_operations rbx1600_audio_fops = {
    open:                rbx1600_audio_open,

```

```

read:      rbx1600_audio_read,
release:   rbx1600_audio_release,
owner:     THIS_MODULE
};

static int audio_dev_id;

static int __init rbx1600_module_init(void)
{
    set_GPIO_mode(GPIO28_BITCLK_I2S_MD);
    set_GPIO_mode(GPIO29_SDATA_IN_I2S_MD);
    set_GPIO_mode(GPIO30_SDATA_OUT_I2S_MD);
    set_GPIO_mode(GPIO31_SYNC_I2S_MD);
    set_GPIO_mode(GPIO32_SDATA_IN1_AC97_MD);

    CKEN |= CKEN8_I2S;
    SACRO = SACRO_RST;
    mdelay(1);
    SACRO = SACRO_ENB | SACRO_RFTH(6) | SACRO_TFTH(6);

    SAIMR = 16; /* enable receive-FIFO interrupts */
    SACR1 = 0; /* I2S operation */

    audio_dev_id = register_sound_dsp(&rbx1600_audio_fops, -1);
    audio_buffer = 0;

    return 0;
}

static void __exit rbx1600_module_exit(void)
{
    unregister_sound_dsp( audio_dev_id );
    SAIMR = 0;
    SACRO = SACRO_RST;
    mdelay(1);
    SACRO = 0;
}

module_init(rbx1600_module_init);
module_exit(rbx1600_module_exit);

MODULE_AUTHOR("Josh Mandel and Keith Winstein");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("I2S Audio Driver for the RBX1600");

EXPORT_NO_SYMBOLS;

```


Sony CD Jukebox Web Server

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>

#define MINVAL(x, y) ((x) < (y) ? (x) : (y))

#define DSP_NAME      "/dev/dsp"
#define SLINK_NAME    "/proc/driver/slink/slc0a"
#define BLOCK_SIZE    588
#define OUTPUT_BLOCK_SIZE 2048
#define SLINK_SIZE    64
#define PORT          80
#define BACKLOG       3
#define QFRAME_MARGIN (4*75*5) /* let playback go 5 seconds over TOC time before cutting it off */
#define BUFF_SIZE     (4096*BLOCK_SIZE)

void write_slink( int fd, char *slink_buffer )
{
    sleep( 1 );
    if ( write( fd, slink_buffer, strlen( slink_buffer ) ) < 0 ) {
        perror( "write s/link" );
    }
}

int serve_request( int disc, int track, FILE *filestar, int fd,
                  int dsp_device, int slink_read, int slink_write,
                  char query_trackcount, char *output_buffer );

void http_error( FILE *filestar, char *error )
{
    fprintf( filestar, "HTTP/1.1 404 Not Found\n" );
    fprintf( filestar, "Content-Type: text/plain\n\n" );
    fprintf( filestar, "%s\n", error );
    fflush( filestar );
}

int sigpipe_received;
void pipe_handler( int signal )
{
    fprintf( stderr, "SIGPIPE received.\n" );
    sigpipe_received = 1;
}

int buffer_contents( int input_i, int output_i )
{
    int contents = input_i - output_i;
    while ( contents < 0 ) {
        contents += BUFF_SIZE;
    }

    return contents;
}

int main( int argc, char **argv )
{
```

```

int disc, track;
int dsp_device, slink_read, slink_write;
int listener, remote;
struct sockaddr_in my_sockaddr;
socklen_t my_sockaddr_len;
FILE *remote_filestar;
struct sigaction pipe_action;
char trackcount_wanted;
char slink_buffer[ SLINK_SIZE ];
char *output_buffer;

output_buffer = (char *) malloc( BUFF_SIZE );
if ( output_buffer == NULL ) {
    fprintf( stderr, "Could not allocate %d bytes of memory\n", BUFF_SIZE );
    return 1;
}

system( "echo 1 > /proc/driver/dac/enable" );
system( "echo 0 > /proc/driver/dac/channel" );
system( "echo 0 > /proc/driver/dac/mute" );

/* don't die on SIGPIPE */

pipe_action.sa_handler = pipe_handler;
sigemptyset( &pipe_action.sa_mask );
pipe_action.sa_flags = 0;

if ( sigaction( SIGPIPE, &pipe_action, NULL ) < 0 ) {
    perror( "sigaction" );
    exit( 1 );
}

if ( (dsp_device = open( DSP_NAME, O_RDONLY )) < 0 ) {
    perror( "open DSP" );
    exit( 1 );
}

/* slink driver has 32-item read FIFO */
if ( (slink_read = open( SLINK_NAME, O_RDONLY | O_NONBLOCK )) < 0 ) {
    perror( "open s/link (read)" );
    exit( 1 );
}

if ( (slink_write = open( SLINK_NAME, O_WRONLY )) < 0 ) {
    perror( "open s/link (write)" );
    exit( 1 );
}

listener = socket( PF_INET, SOCK_STREAM, 0 );
if ( listener < 0 ) {
    perror( "socket" );
    exit( 1 );
}

my_sockaddr.sin_family = AF_INET;
my_sockaddr.sin_addr.s_addr = INADDR_ANY;
my_sockaddr.sin_port = htons( PORT );

if ( bind( listener,
          (struct sockaddr *)&my_sockaddr, sizeof( my_sockaddr ) ) < 0 ) {
    perror( "bind" );
    exit( 1 );
}

while ( 1 ) {

```

```

if ( listen( listener, BACKLOG ) < 0 ) {
    perror( "listen" );
    exit( 1 );
}

my_sockaddr_len = sizeof( my_sockaddr );

remote = accept( listener, (struct sockaddr *)&my_sockaddr,
                &my_sockaddr_len );
if ( remote < 0 ) {
    perror( "accept" );
    close( remote );
    continue;
}

sigpipe_received = 0;

remote_filestar = fdopen( remote, "r+" );
if ( remote_filestar == NULL ) {
    perror( "fdopen" );
    exit( 1 );
}

disc = 0;
track = 0;
trackcount_wanted = 0;

while( 1 ) {
    char line_buffer[ 2048 ];

    if ( sigpipe_received ) {
        break;
    }

    if ( fgets( line_buffer, 2048, remote_filestar ) == NULL ) {
        break;
    }

    if ( (strcmp( line_buffer, "\r\n" ) == 0)
        || (strcmp( line_buffer, "\r" ) == 0)
        || (strcmp( line_buffer, "\n" ) == 0) ) {
        break;
    }

    if ( strstr( line_buffer, "trackcount" ) != NULL ) {
        trackcount_wanted = 1;
    }

    if ( strncmp( line_buffer, "GET ", 4 ) == 0 ) {
        strtok( line_buffer, "/" );
        disc = atoi( strtok( NULL, "/" ) );
        track = atoi( strtok( NULL, "/" ) );
    }
}

if (!sigpipe_received) {
    serve_request( disc, track, remote_filestar, remote,
                  dsp_device, slink_read, slink_write, trackcount_wanted,
                  output_buffer );
}

fclose( remote_filestar );

/* stop */
snprintf( slink_buffer, SLINK_SIZE, "109001\n" );

```

```

    write_slink( slink_write, slink_buffer );
}

fprintf( stderr, "Quitting.\n" );
return 1;
}

int serve_request( int disc, int track, FILE *filestar, int fd,
                  int dsp_device, int slink_read, int slink_write,
                  char query_trackcount, char *output_buffer )
{
    int bytes_read, bytes_written_this_time;
    int cur_disc, cur_track;
    int control_b, disc_a, disc_b;
    char buffer[ BLOCK_SIZE ];
    char slink_buffer[ SLINK_SIZE ];
    char hit = 0;
    int total_quarter_frames = 0;
    int read_quarter_frames = 0;
    int input_i = 0, output_i = 0;
    char input_eof = 0, buffer_full = 0, buffer_empty = 1;

    if ( (disc < 1) || (disc > 400) ) {
        http_error( filestar, "Only 400 discs are supported." );
        return 1;
    }

    if ( (track < 1) || (track > 99) ) {
        http_error( filestar, "Only 99 tracks are supported." );
        return 1;
    }

    cur_disc = 0;
    cur_track = 0;

    fcntl( fd, F_SETFL, O_NONBLOCK );

    /* power on */
    snprintf( slink_buffer, SLINK_SIZE, "10902e\n" );
    write_slink( slink_write, slink_buffer );
    if ( sigpipe_received ) { return 1; }

    /* lock front panel */
    snprintf( slink_buffer, SLINK_SIZE, "109020\n" );
    write_slink( slink_write, slink_buffer );
    if ( sigpipe_received ) { return 1; }

    /* stop */
    snprintf( slink_buffer, SLINK_SIZE, "109001\n" );
    write_slink( slink_write, slink_buffer );
    if ( sigpipe_received ) { return 1; }

    /* flush s/link buffer */
    sleep( 1 );
    while( 1 ) {
        if ( sigpipe_received ) { return 1; }
        if ( read( slink_read, slink_buffer, SLINK_SIZE ) < 0 ) {
            break;
        }
    }
}

if ( sigpipe_received ) { return 1; }

/* cue track 1 */
if ( disc <= 99 ) {

```

```

    control_b = 0;
    disc_a = disc / 10;
    disc_b = disc % 10;
} else if ( disc <= 200 ) {
    control_b = 0;
    disc_a = ((disc+54) & 0xf0) >> 4;
    disc_b = (disc+54) & 0x0f;
} else {
    control_b = 3;
    disc_a = ((disc-200) & 0xf0) >> 4;
    disc_b = (disc-200) & 0x0f;
}

snprintf( slink_buffer, SLINK_SIZE, "309%1d51%x%x01\n",
          control_b, disc_a, disc_b );
// fprintf( stderr, "Sending: %s", slink_buffer );
write_slink( slink_write, slink_buffer );

/* wait for ready */
while( 1 ) {
    if ( sigpipe_received ) {
        return 1;
    }

    memset( slink_buffer, 0, SLINK_SIZE );
    if ( read( slink_read, slink_buffer, SLINK_SIZE ) > 0 ) {
        // fprintf( stderr, "Got: %s", slink_buffer );

        /* check for play event */
        if ( memcmp( slink_buffer + 4, "50", 2 ) == 0 ) {
            int cur_control_b, cur_disc_a, cur_disc_b;
            int cur_track_a, cur_track_b;
            sscanf( slink_buffer + 3, "%1x", &cur_control_b );
            sscanf( slink_buffer + 6, "%1x%1x%1d%1d",
                    &cur_disc_a, &cur_disc_b, &cur_track_a, &cur_track_b );

            if ( cur_control_b != control_b + 8 ) {
                http_error( filestar, "Unrecognized control code." );
                return 1;
            }

            if ( cur_control_b == 11 ) {
                cur_disc = cur_disc_a * 16 + cur_disc_b + 200;
            } else if ( cur_control_b == 8 ) {
                if ( cur_disc_a > 9 ) {
                    cur_disc = 16 * cur_disc_a + cur_disc_b - 54;
                } else {
                    cur_disc = 10 * cur_disc_a + cur_disc_b;
                }
            } else {
                http_error( filestar, "Unrecognized control code." );
                return 1;
            }
            cur_track = cur_track_a * 10 + cur_track_b;

            if ( (cur_disc != disc) || (cur_track != 1) ) {
                http_error( filestar, "No such disc." );
                return 1;
            }

            break;
        }
        if ( memcmp( slink_buffer + 4, "53", 2 ) == 0 ) {
            http_error( filestar, "No such disc." );
            return 1;
        }
    }
}

```

```

    }
    if ( memcmp( slink_buffer + 4, "05", 2 ) == 0 ) {
        http_error( filestar, "No such disc." );
        return 1;
    }
}
}

/* stop */
snprintf( slink_buffer, SLINK_SIZE, "109001\n" );
write_slink( slink_write, slink_buffer );
if ( sigpipe_received ) { return 1; }

/* query disc info */
snprintf( slink_buffer, SLINK_SIZE, "109%1d44\n", control_b );
write_slink( slink_write, slink_buffer );
if ( sigpipe_received ) { return 1; }

/* get track count */
while( 1 ) {
    if ( sigpipe_received ) {
        return 1;
    }

    memset( slink_buffer, 0, SLINK_SIZE );
    if ( read( slink_read, slink_buffer, SLINK_SIZE ) > 0 ) {
        // fprintf( stderr, "Got: %s", slink_buffer );

        /* check for 0x14 */
        if ( memcmp( slink_buffer + 4, "14", 2 ) == 0 ) {
            http_error( filestar, "Disc not loaded." );
            return 1;
        }

        /* check for 0x60 */
        if ( memcmp( slink_buffer + 4, "60", 2 ) == 0 ) {
            int tracks_a, tracks_b;
            sscanf( slink_buffer + 10, "%1d%1d", &tracks_a, &tracks_b );

            int total_tracks = tracks_a * 10 + tracks_b;

            if ( query_trackcount ) {
                fprintf( filestar, "HTTP/1.1 200 OK\n" );
                fprintf( filestar, "Content-Type: text/plain\n\n" );
                fprintf( filestar, "Disc %d has %d tracks.\n", disc, total_tracks );

                fflush( filestar );
                return 1;
            }

            if ( track > total_tracks ) {
                http_error( filestar, "No such track." );
                return 1;
            }
        }
        break;
    }
}

if ( sigpipe_received ) {
    return 1;
}

/* request track */
snprintf( slink_buffer, SLINK_SIZE, "309%1d50%1x%1x%d%d\n",
        control_b, disc_a, disc_b, (track / 10), (track % 10) );

```

```

// fprintf( stderr, "Sending: %s", slink_buffer );
write_slink( slink_write, slink_buffer );

while( 1 ) {
    if ( sigpipe_received ) {
        return 1;
    }

    memset( slink_buffer, 0, SLINK_SIZE );
    if ( read( slink_read, slink_buffer, SLINK_SIZE ) > 0 ) {
        //      fprintf( stderr, "Got: %s", slink_buffer );

        if ( memcmp( slink_buffer + 4, "53", 2 ) == 0 ) {
            http_error( filestar, "No such disc." );
            return 1;
        }
        if ( memcmp( slink_buffer + 4, "05", 2 ) == 0 ) {
            http_error( filestar, "No such disc." );
            return 1;
        }
        }

    /* check for stop event */
    if ( memcmp( slink_buffer + 4, "01", 2 ) == 0 ) {
        if ( hit ) {
            input_eof = 1;
        } else {
            http_error( filestar, "Stopped." );
            return 1;
        }
    }

    /* check for play event */
    if ( memcmp( slink_buffer + 4, "50", 2 ) == 0 ) {
        int cur_control_b, cur_disc_a, cur_disc_b;
        int cur_track_a, cur_track_b;
        int track_min, track_sec;

        sscanf( slink_buffer + 3, "%1x", &cur_control_b );
        sscanf( slink_buffer + 6, "%1x%1x%1d%1d",
                &cur_disc_a, &cur_disc_b, &cur_track_a, &cur_track_b );
        sscanf( slink_buffer + 10, "%2d%2d",
                &track_min, &track_sec );

        if ( cur_control_b != control_b + 8 ) {
            http_error( filestar, "Unrecognized control code." );
            return 1;
        }

        if ( cur_control_b == 11 ) {
            cur_disc = cur_disc_a * 16 + cur_disc_b + 200;
        } else if ( cur_control_b == 8 ) {
            if ( cur_disc_a > 9 ) {
                cur_disc = 16 * cur_disc_a + cur_disc_b - 54;
            } else {
                cur_disc = 10 * cur_disc_a + cur_disc_b;
            }
        } else {
            http_error( filestar, "Unrecognized control code." );
            return 1;
        }
        cur_track = cur_track_a * 10 + cur_track_b;

        if ( (cur_disc == disc) && (cur_track == track) ) {
            hit = 1;
        }
    }
}

```

```

total_quarter_frames = track_sec * 75 * 4 + track_min * 60 * 75 * 4;
read_quarter_frames = 0;

fprintf( filestar, "HTTP/1.1 200 OK\n" );
fprintf( filestar, "X-Track-Length: %d minutes, %d seconds\n", track_min, track_sec );
fprintf( filestar, "X-Track-Quarter-Frames: %d\n", total_quarter_frames );
fprintf( stderr, "Playing disc %d, track %d (%d minutes, %d seconds).\n", disc, track,
        track_min, track_sec );
//      fprintf( filestar, "Content-Length: %d\n", (total_quarter_frames + QFRAME_MARGIN ) * BLOCK_SIZE );
// Do not send almost-correct content length or client will retry!
fprintf( filestar, "Content-Type: audio/x-16bit-be-signed-stereo\n\n" );
fflush( filestar );
} else {
/* stop playback */
snprintf( slink_buffer, SLINK_SIZE, "109001\n" );
write_slink( slink_write, slink_buffer );

if ( hit ) {
    input_eof = 1;
} else {
    http_error( filestar, "No such track." );
    return 1;
}
}
}

if ( hit ) {
if ( sigpipe_received ) { return 1; }

if ( buffer_contents( input_i, output_i ) == 0 ) {
    buffer_empty = 1;
} else {
    buffer_empty = 0;
}

if ( buffer_contents( input_i, output_i ) > BUFF_SIZE - 2*BLOCK_SIZE ) {
    buffer_full = 1;
    fprintf( stderr, "Buffer full.\n" );
} else {
    buffer_full = 0;
}

if ( buffer_empty && input_eof ) {
    return 0;
}

bytes_read = read( dsp_device, buffer, BLOCK_SIZE );
if ( bytes_read <= 0 ) {
    bytes_read = read( dsp_device, buffer, BLOCK_SIZE );
    if ( bytes_read <= 0 ) {
        perror( "read DSP (x2)" );
        return 1;
    }
}

read_quarter_frames++;
if ( read_quarter_frames > total_quarter_frames + QFRAME_MARGIN ) {
    fprintf( stderr, "Time expired on disc %d, track %d. (read: %d, TOC: %d)\n", disc, track,
            read_quarter_frames, total_quarter_frames );
    input_eof = 1;
}

if (!input_eof) {
    if ( BUFF_SIZE - input_i >= bytes_read ) {

```