Voice-Over-IP Final Report

April 25, 2000

Project Members:

Todd Stokes
David Jeffery
Quiana Smith
David Ferguson

Georgia Institute of Technology

ECE4005

Professor: Dr. Henry Owen

Table of Contents

1. Abstract

With the rising power of the Internet, the concept of moving voice over data packet networks has emerged as a viable technology for the future. The basics of a voice-over-IP, or VoIP, system require programs running on the two endpoint computers that translate voice into packetized data and vice-versa. The data is shipped across a data link fast enough for real-time communication. This basic VoIP system is implemented in our project.

2. Objectives

We intend to build a VoIP system between two computers linked by a 10-megabit network. When finished, we will demonstrate full-duplex communication between the two computers. Our program, which will run on both computers, will establish a socket data link with the other side. The program will simultaneously write and read data to and from the sound card, while handling the packetizing and transmission of data across the socket layer. A simple GUI will be included to make the application more user-friendly. If time permits, audio processing may be added to compress the transmitted data and to better handle lost or corrupted data.

2.1 Materials Needed

For implementation, the VoIP project required the following hardware and software components:

- Two computers with two hard drives
- Two sound cards that support full-duplex operation
- Working network (with Internet access if possible)
- RedHat Linux 6.1 CD
- Two sets of speakers and microphones

Note that since we used Linux as our chosen platform, the sound cards must be supported by the Advanced Linux Sound Architecture (ALSA) project.

2.2 Challenges and Anticipated Problems

The first challenge our group had was getting our less Linux-knowledgeable members up to speed on the platform being used. A good base knowledge of the applications available for Linux and within different distributions of Linux can be hard to come by. After installing Linux on the computers in the lab and some of our machines at home, the lack of familiarity with Linux programming was less of a problem than anticipated.

The first problem run into in the lab was the inability of the Vibra soundcards to do duplex sound. While the cards are capable of full duplex, the specifications have never

been released by Creative Labs so the ALSA project has no way of implementing a driver that can use full duplex. Other than this problem with the Vibra cards, installation and use of ALSA has gone smoothly. With the new PCI Soundblasters installed to replace the Vibras, full duplex sound has worked smoothly. Linux sound support is not an issue with the newer cards.

There is a problem with the original sound code that generates a very noticeable delay between recording and playback. It was caused by a problem with the way ALSA handles date exchange when using file IO. ALSA would not start playing sound until the entire buffer was filled. Since it would take approximately two seconds to fill the entire buffer, it resulted a very obvious and annoying delay. The first attempted solution was to alter the program to use memory mapped IO instead. This by itself did not fix the problem. ALSA has a bug in it that prevents it from continuously playing when the buffer is not completely full. The solution was to force ALSA in to a continuous loop mode. When using this mode, ALSA continuously loops through its sound buffer and plays the data held there regardless of whether or not the data is new or old. Using this trick to play sound forces the VoIP program to make sure it is always inserting sound before the part of the buffer ALSA is playing. If the program doesn't, ALSA starts playing old data and in effect start playing garbage.

The use of memory mapped IO made managing the sound buffer easier. However, using memory mapped IO is more complicated and requires more interaction and intervention than the simple and direct file reads and writes. The added complexity of mmap IO is a price that must be paid to insure minimal sound delay. Example code for using mmap with ALSA has been found from two sources: aplay and quakeforge. Aplay, from the alsa-utils package at www.alsa-project.org, is ALSA's example program for both playing and recording. Quakeforge is an open source computer game, found at quake.sourceforge.net, that uses mmap IO with ALSA sound to play sound effects with no perceivable delay.

The basic network sockets code was completed without any major problems. The problems didn't begin until an actual protocol was implemented. Since the VoIP program uses UDP, the program had to be capable of dealing with lost packets. The unreliability of UDP created problems with ensuring that clients would receive the connection requests and confirmation packets. The solution to getting the initial request was for the connecting client to repeatedly send its request until it gets an answer or the request is canceled. The nastier problem was insuring the connecting client would get the confirmation information that contained the format and rate the data would be transmitted in. Instead of having a special packet to return this information, the format and rate are "piggybacked" in the header of the data packet. This way, once the client gets a data packet it has an easy way of finding out the format and rate to use.

2.3 Example Commercial or Open-Source Implementations

2.3.1 Speak Freely Release 7.1

Speak Freely is an application used on a variety of Unix workstations that allows for talking (a sent voice, not typed characters) over a network. If your network connection is too slow to support real-time voice data, several forms of compression may allow you, assuming your computer is fast enough, to converse nonetheless. Encryption with DES, Blowfish, IDEA, and/or a key file is available to enable secure communications. If PGP is installed on the user's machine, it can be called upon automatically to exchange IDEA session keys for a given conversation. Speak Freely for Unix is compatible with Speak Freely for Windows, and users of the two programs can intercommunicate.

Release 7.1 focuses on specific problems with audio drivers on several platforms, in particular Linux. A number of additional configuration parameters, documented in the makefile, allow for the configuration of Speak Freely for the wide variety of audio drivers one encounters in Linux systems. A new sflaunch program permits running Speak Freely on systems that do not allow two independent programs to open the audio device, even if one does so read-only and the other write-only. Speak Freely for Unix is documented in the following manual pages describing the individual programs.

sfmike Send sound to remote hosts(s).
sfspeaker Receive sound.
sflaunch Launch sfmike and sfspeaker.
sflwl Find active users on the network.
sflwld Operate a find-users server.
sfecho Operate a remote echo server.
sfreflect Operate conference reflector.
sfvod Operate a voice-on-demand server.

2.4 Freely Available Tools and APIs

2.4.1 ALSA

A helpful website for design of the sound code of the project was found at www.alsa-project.org. This is an organization of Linux developers called ALSA who make sound drivers, a sound card API, and a set of sound utilities available, all open-source, for other Linux developers. Unfortunately, the documentation for the ALSA API was very poor at the start of the VoIP project.

While the ALSA project writes quality code, the ALSA documentation at the time the VoIP project started was worthless. In addition to being old, it was horribly wrong after they have made several interface changes. The only redemption of the old documentation is that it gave enough of a general idea of how ALSA works, that one could read the

source code for the sample player and recorder that comes with the alsa-utils package to understand how to program to ALSA's interfaces.

Recently, ALSA finally posted updated documentation to the ALSA API. The new documentation, available at www.alsa-project.org/documentation.php3 , corrects the inaccuracies found in the old documentation and would have made programming the sound code much easier. However, the new documentation was not in time to be useful in the programming of the VoIP program.

While ALSA's library documentation was lacking, ALSA does have a very complete installation and setup HOWTO available on their web site. The ALSA HOWTO is located on the same page as the new ALSA API documentation.

2.4.2 KDOC

KDOC is a C++ and IDL (interface definition language) interface documentation tool, initially written in order to generate documentation for the KDE libraries. KDOC removes specially formatted documentation and information about your classes from the class' header or IDL files, and creates cross-referenced HTML, LaTeX or Man pages from it. KDOC permits groups of classes to be formed into "libraries" and documentation from separate libraries can be cross-referenced very easily. This documentation tool supports Qt signals and slots. KDOC is written in PERL and is easily extensible.

2.5 Choosing the GUI Toolkit

2.5.1 GUI Toolkit Tradeoffs

Since the group has no experience with programming GUI's in Linux, we looked at various GUI toolkits that were available. A GUI toolkit simply wraps the low-level X-Windows drawing calls and presents a nice Application Programming Interface (API) to the applications programmer. A GUI toolkit also offers "widgets." Widgets encapsulate a given GUI functionality and may be placed in a larger framework to create a complete GUI. For instance, a button or a listview is a widget. Different GUI toolkits offer varying degrees of encapsulation and reuse. Most GUI toolkits also have development tools that allow for "drag-and-drop" creation of widgets and dialog boxes.

We looked at three toolkits, GTK+, Qt, and Tcl/Tk. Tcl/Tk is based on the scripting language, Tcl. Rather than learn a new language, we focused mainly on the C/C++ based solutions, GTK+ and Qt.

GTK+ has wide acceptance and support in the Linux community. It also has an excellent development tool called Glade. Glade seems very stable and feature rich. However, since GTK+ is based on standard C, callbacks and other tricks must be used to implement object-oriented widgets. Callbacks, being weakly typed, are notorious for causing

segmentation faults. In the end, GTK+ seemed too cumbersome to implement clean GUI code.

Qt also has a large support base, as it is the GUI toolkit of choice for the KDE desktop environment. Qt is based on C++, allowing for much more object-oriented code. Qt follows a hierarchical, inheritance-based model for GUI implementation. For instance, the programmer may create a widget that contains many other widgets. He or she may implement a complete functionality in this larger widget and then may place it in a super widget along with other widgets to combine more and more encapsulated functionalities into an application. Because it is written in C++, Qt does not have a reliance on callbacks and is, therefore, a much more strongly typed implementation.

Unfortunately, the development tool for Qt, called KDevelop, is currently in beta testing. As a result, not all widgets are available in the "drag-and-drop" dialog creator. In addition, the program, as a whole, has some bugs that will cause crashes or unexpected behavior. On the other hand, KDevelop does have many good features that make using it worthwhile. CVS version control software is built into the development environment. CVS is especially good at managing projects where developers are working from remote locations, as we are. This CVS integration is very important to easing the task of project management. Furthermore, KDevelop handles all the make scripts, including making of the documentation and distribution. Little to no interaction is necessary to create and maintain project building.

After weighing the costs and benefits of the three GUI toolkits, we chose Qt for its object-oriented implementation, its highly integrated development tool, and its ease of use. Qt may be found at http://www.troll.no/qt, and the homepage for KDevelop is http://www.kdevelop.org. Since Qt 1.44 comes standard on RedHat 6.0 and 6.1, the Qt installation was already complete. Installation of KDevelop was also relatively easy, even though the KDevelop application must be compiled on the local machine. The difficult part was the KDevelop setup, since KDevelop requires around 10 external applications, such as cvs, glimpse, and kdoc, for full operation. RedHat does install some of these by default, but the user must track down and successfully install a few. The KDevelop user's manual, found on the KDevelop home page, describes where to find these required external applications.

2.5.2 KDevelop Installation Summary

Kdevelop is available from the KDE Developer's Kit, or KDK. To obtain the KDK source package, open http://www.kdevelop.org and click on "Download." This will bring up a list of the FTP sites where the KDK is available. Assuming that KDK 1.1 is the newest release, download the file, "kdk-1.1.tar.gz." From a root shell, enter the following commands:

```
% tar xzvf kdk-1.1.tar.gz
% cd kdk-1.1
% ./configure
```

```
% make
% make install
```

If configure complains about a wrong version of Qt, check to make sure that you have the environmental variable QTDIR pointing to a 1.4x version. On RedHat 6.1, QTDIR should be /usr/lib/qt-1.44.

Once installed, Kdevelop may be launched by typing, “kdevelop,” at a shell prompt.

2.6 Project Planning and Logistics

The VoIP program code consists of four basic building blocks: the sound backend, the network sockets backend, the GUI control interface, and the system control routines. The sound backend will be responsible for handling communication with the soundcard. The network sockets backend will be responsible for sending and receiving the data as it is sent across the network. The GUI control interface is responsible for allowing the user to control and initiate communication. The system control routines are responsible for tying the GUI and backends together into one cohesive whole.

2.6.1 Tasks and Division of Labor

Task	**Group Members**
Sound Backend	David Jeffery Todd Stokes
Network Sockets Backend	Todd Stokes Quiana Smith David Jeffery
GUI Control Interface	David Ferguson Quiana Smith
Connection State Machine	David Ferguson David Jeffery

2.6.2 CVS

To address the source integrity challenges posed by four project members working on the same code, a simple CVS server was set up on one of our Linux boxes at home. Getting CVS usable by everyone was a challenge as the CVS program can be a hassle to use correctly and efficiently. Another key strategy for source integrity was in the division of object code. Dividing the code into proper sections minimizes the likelihood of new code interfering with the work of someone else’s code.

CVS should come with every RedHat 6.0/6.1+ installation. Make sure you have the CVS rpm installed by executing "rpm -q -a |grep cvs" If nothing comes up, you'll have to

download the CVS rpm from a RedHat mirror, like ftp.cc.gatech.edu/linux/distributions/redhat.

Comprehensive documentation for installation and use of CVS for Linux can be found at http://www.gnu.org/manual/cvs/index.html. There is also a lot of information about general source integrity issues and a lot of helpful development tips. What follows is a brief summary of the server setup process, login, and the most commonly used commands.

2.6.2.1 Setting up a CVS Server

To set up a CVS repository, choose a directory with ample disk space available for the revision history of the source files. It should be accessible (directly or via a networked file system) from all machines that want to use CVS. It is not possible to use CVS to read from a repository which one only has read access to; CVS needs to be able to create lock files.

To create a repository:

1. Run the `cvs -d` *`repository_pathname`* `init` command. It will set up an empty repository in the CVS root specified in the usual way. `cvs init` is careful to never overwrite any existing files in the repository, so no harm is done if you run `cvs init` on an already set-up repository.

2. `cvs init` will enable history logging; if you don't want that, remove the history file after running `cvs init`.

2.6.2.2 Logging In to a CVS Server

To use CVS (once it has been successfully installed):

1. Set the environmental variable CVSROOT to ":pserver:*username@host.org*:/*repository_pathname*". In bash, type `export CVSROOT=:pserver:`*`username@host.org`*`:/repository_pathname`
2. Make sure that you have CVSROOT set correctly and execute "cvs login"
3. Enter your password
4. If all goes correctly, it should say nothing.

2.6.2.3 Some Basic CVS commands

- `cvs commit` - incorporates changes from your working source files into the source repository (user is prompted to type a log message)

- `cvs update` - reconciles your work with any revisions applied to the source repository since your last checkout or update (CVS will alert you if anything you have edited may be lost in the update)
- `cvs checkout` - recursively creates directories and populate them with the appropriate source files
- `cvs diff` - allows you to see the differences between two revisions of a file
- `cvs log` - shows you all of the commit log entries that are associated with a file

2.6.3 Project Timeline Comparison

Date	Intended Schedule	Difficulty	Actual Schedule
Week 4	Read from Microphone	Medium	Read from Microphone
	Output sound to speakers	Medium	Output sound to speakers
	Examine/study GUI tools	Easy	Examine/study GUI tools
	Start socket code	Easy	Start socket code
Week 5	Read from microphone and play back through speakers	Medium/ Hard	Read from microphone and play back through speakers
	Send random data over sockets between machines	Easy	Send random data over sockets between machines
	Design basic GUI controls	Medium	Design basic GUI controls
Week 6	Design or reuse a protocol	Easy	
	Start abstraction layer between GUI and backend	Medium	Start abstraction layer between GUI and backend
	(Milestone 1) Send and play sound data over sockets	Hard	***(Milestone 1)***
Week 7	Integrate protocol into sockets code	Hard	Design a protocol
	Merge all code into one code base	Very Hard	Merge all code into one code base
Week 8	Integrate GUI and backend into one complete program	Very Hard	Solved Sound Delay Problem
	(Milestone 2) Send sound at different rates	Medium	Integrate protocol into sockets code
Week 9	***(Milestone 3)*** Use GUI to control backend	Hard	***(Milestone 2)***
Week 10	***(Milestone 4)*** Carry conversation between two people	Medium	Integrate GUI and backend into one complete program ***(Milestone 3)***
Week 11	Feasibility of additional options:	N/A	***(Milestone 4)***

	• IPv6 • Compression • Additional backends • Encryption • Teleconference		
Week 12	Begin integrating any new additions	Medium	Coordinated GUI with the backend. Solved Sound Card Release Problem.
Week 13	Test new additions	Medium	Added ability to Force Sampling Rate for better Sound Quality. Solved some crashes on Disconnect/Reconnect
Week 14	***(Milestone 5)*** Carry conversations with all allowed combinations of options	Medium	Finished Testing and Debugging Final Product. Paper and Presentation. ***(Milestone 5)***

3. KVOIP Technical Explanation

3.1 Sound Backend

The KVoIP sound code is responsible for playing sound, recording sound, and checking for valid formats and rates during sound format negotiation. The sound code uses the ALSA sound library to interface with the sound card. At the start of KVoIP, the allowed formats and rates are retrieved from ALSA and saved. Data is copied to and from the sound buffers using memory mapped devices. To allow communication between more types of hardware, software sound conversions are also handled seamlessly without the rest of KVoIP being aware of conversions being used.

3.2 Network Sockets Backend

The KVoIP networking supplies the basic building blocks needed to run the KVoIP VoIP protocol. It handles the sending and receiving of packets. It provides methods for binding a port and establishing a connection to another host. The basic tools needed to construct the protocol headers are supplied. The networking code also provides a simple, complete method for tearing down an established connection.

3.2.1 The KVOIP Network Protocol

Control	Request	Pad	Pad
Sound Rates			
Sound Formats			

KVoIP's first header format is used for communicating requests between clients. Open and close requests use this header. The sound rates and formats are supplied on open requests. These two header fields are ignored on close requests.

Data	Rate #	Format #	Pad
Identity Counter			
Sound Data			

KVoIP's second header format is used for carrying the sound data. It also supplies the rate and format the sound is transmitted in and a counter to insure the sound data is played in the correct order.

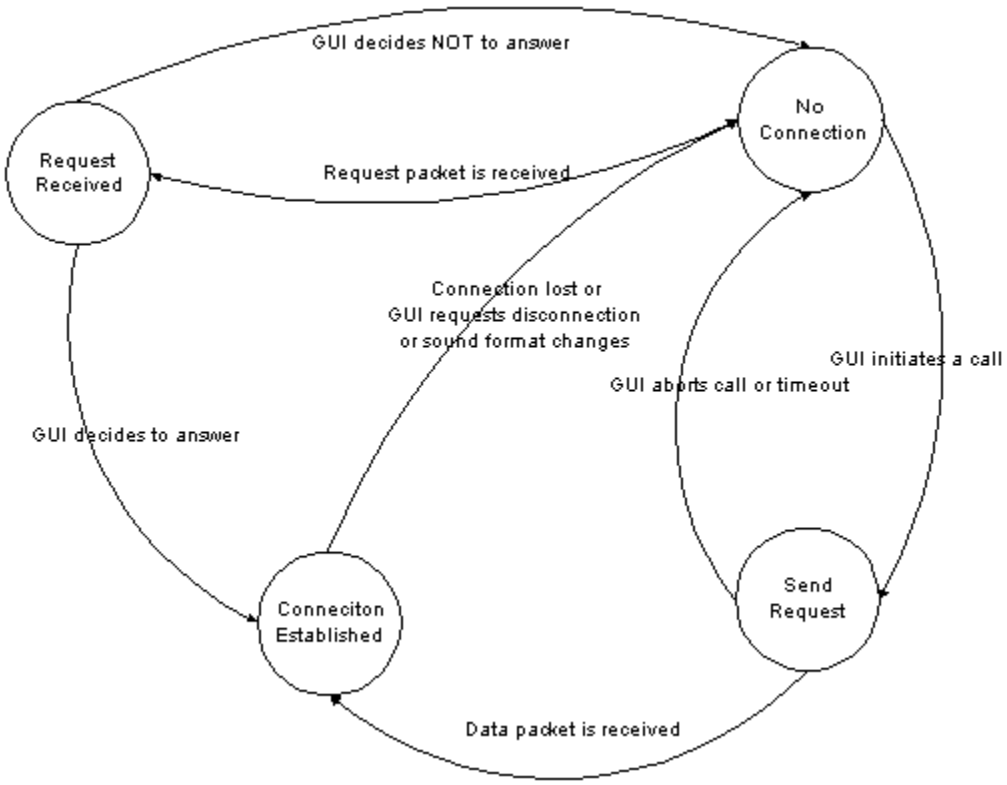
3.3 Connection State Machine

The Connection State Machine, or CSM, creates the link between the GUI and the low-level backend. The CSM handles incoming calls and negotiating outgoing calls. Within KvoIP, no routine is allowed to block, waiting for user input or some I/O device to become available. Otherwise, the GUI would lock up or become generally unresponsive. Through Qt's signal and slot architecture, the CSM is able to monitor the GUI and the socket file descriptor in a notification based manner. This allows the GUI to continue servicing user interaction while the CSM responds to network activity.

Once the CSM has successfully connected to another KVoIP, control is handed over to the backend. The CSM ceases listening on the socket and waits for the backend to finish (for whatever reason).

See the figure below for the state diagram.

Figure 3.1 State diagram for the Connection State Machine (CSM)



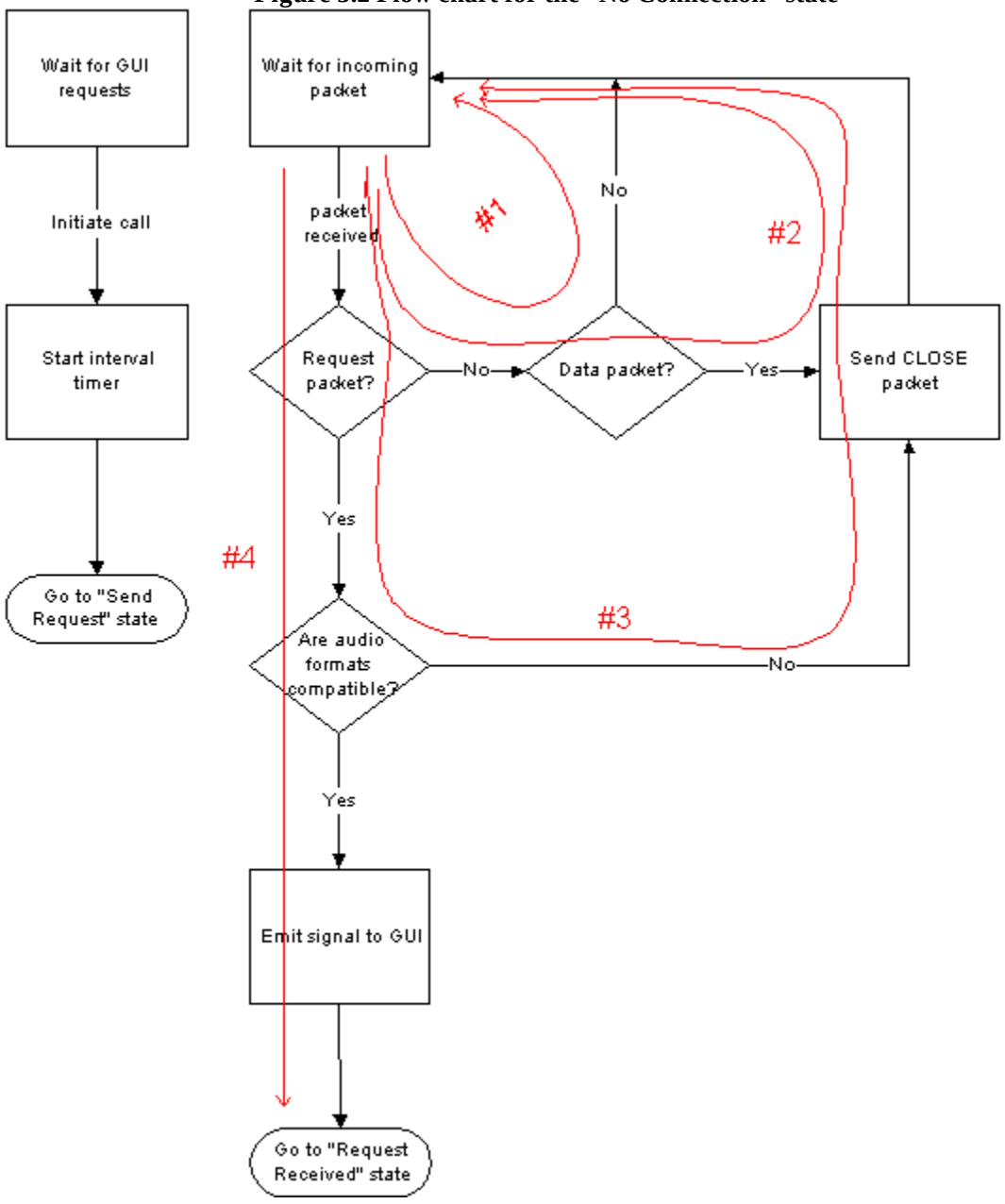
Within each state, the CSM makes decisions that affect the backend and the GUI.

3.3.1 The "No Connection" State

In the "No Connection" state, KVoIP is not connected to another computer and the CSM is listening for GUI commands and for incoming network packets.

See the figure below for the "No Connection" flow chart.

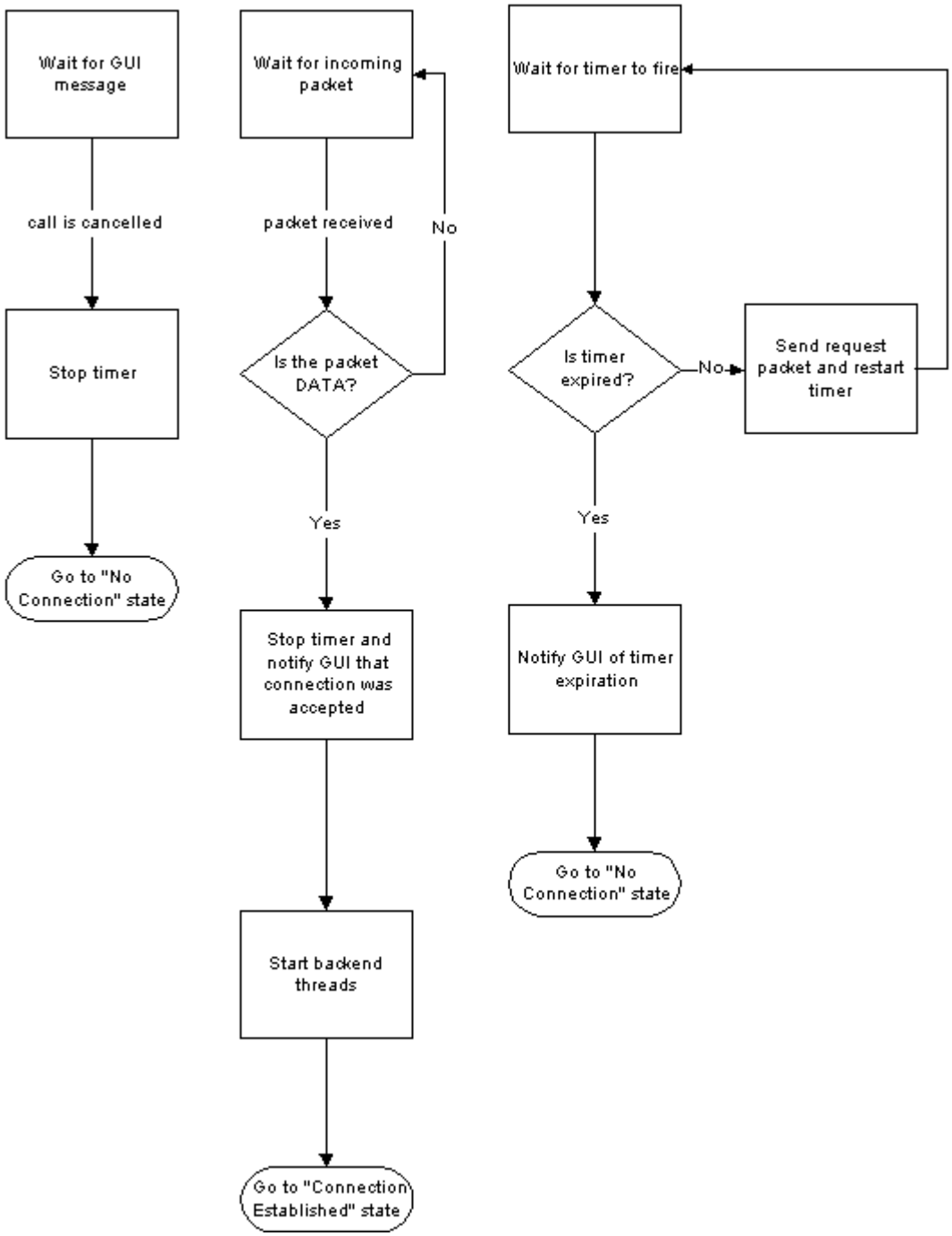
Figure 3.2 Flow chart for the "No Connection" state



3.3.2 The "Send Request" State

In the "Send Request" state, the GUI has already told the CSM to contact another host for a call. Once every second, the CSM sends out a request packet to the remote computer. The GUI can either cancel the call or the remote computer can reject or accept the request.

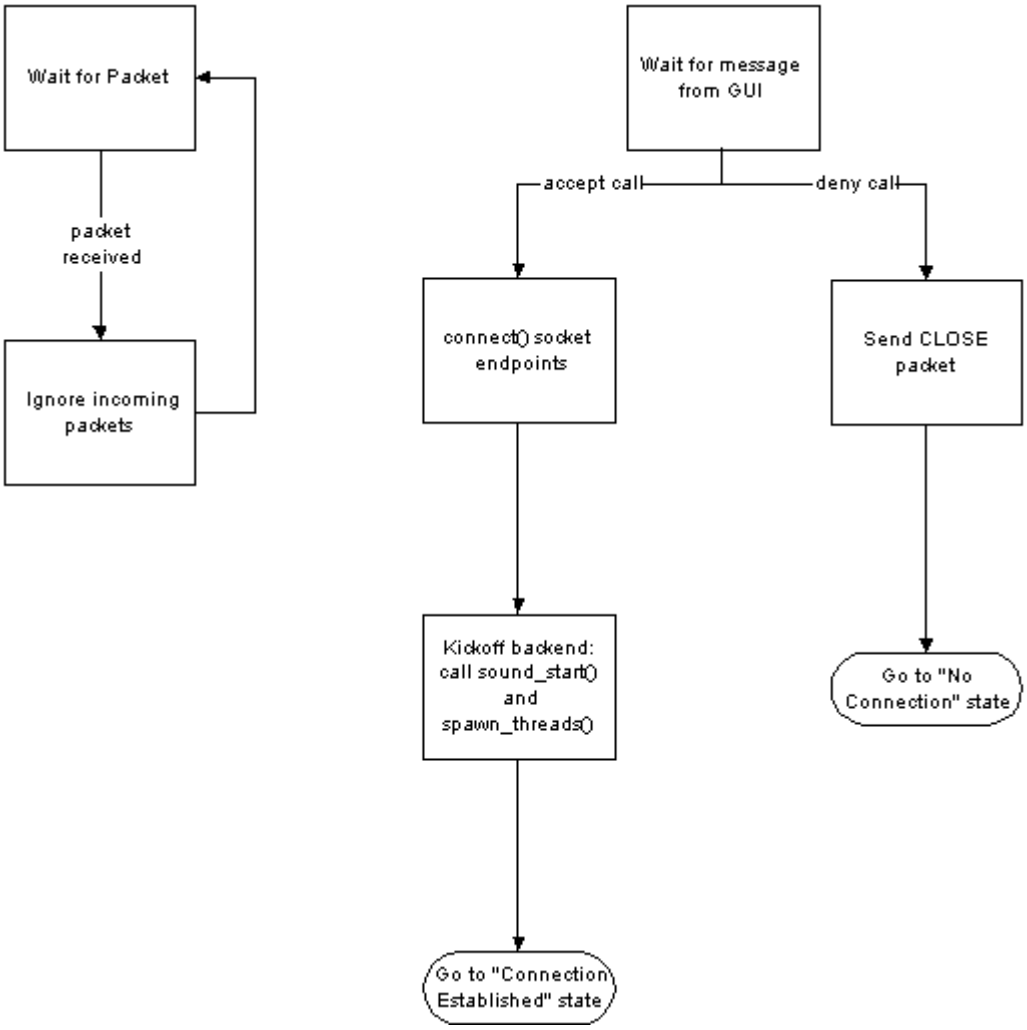
Figure 3.3 Flow chart for the “Send Request” state



3.3.3 The “Request Received” State

In the “Request Received” state, the CSM has already received a request packet from a remote computer. The CSM has placed a dialog box on the GUI, asking the user if he or she wishes to accept or reject the incoming call. In this state, the CSM will ignore incoming packets and will only advance once the user has made his decision.

Figure 3.4 Flow chart for the “Request Received” state



3.3.4 The “Connection Established” State

The “Connection Established” state is a special state where the CSM has already started and given control over to the backend. The CSM simply waits for the backend to die. The backend can die for a variety of reasons, such as a network error or the other end

hung up. In this state, a timer alerts the CSM to check the state of the backend and go back to sleep if it is still running. During this check, the CSM also notifies the GUI to update its network statistics and audio properties.

3.4 Design of the GUI

The design of the KVoIP GUI is very simple. Its job is to provide an easy way for connections to be established and to display some feedback about the network quality and the audio stream properties.

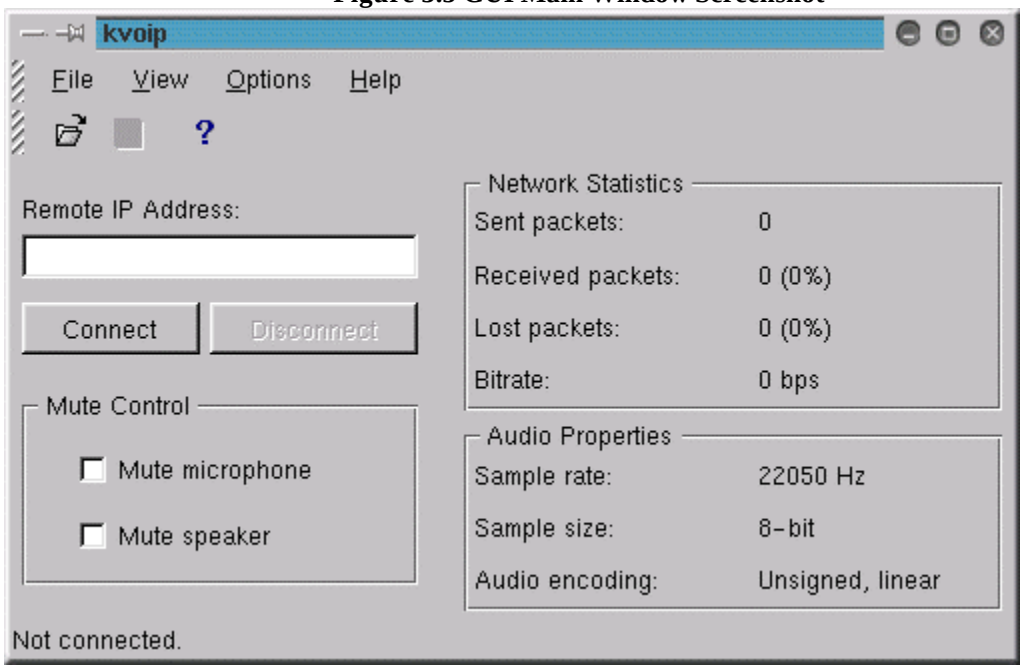
KVoIP GUI features:

- Quick entry connection/disconnection buttons
- Mute control for muting the microphone or speaker or both
- Status bar with the current status of the connection
- Dynamically updated network statistics and audio properties are shown on the right.
- Full featured address book for connections to hard to remember addresses
- Recently connected addresses appear in the File menu for quick connects
- Full on-line HTML help accessible from the Help menu
- Draggable toolbar and menubar
- Fully resizable main window
- All settings are saved, including window size, recent hosts, address book data, maximum sample rate, and mute configuration.

Network statistics are displayed on the right. The user can use these to see if packets are being sent or if lost packets are causing sound quality problems.

See the figure below for a screenshot of the main window

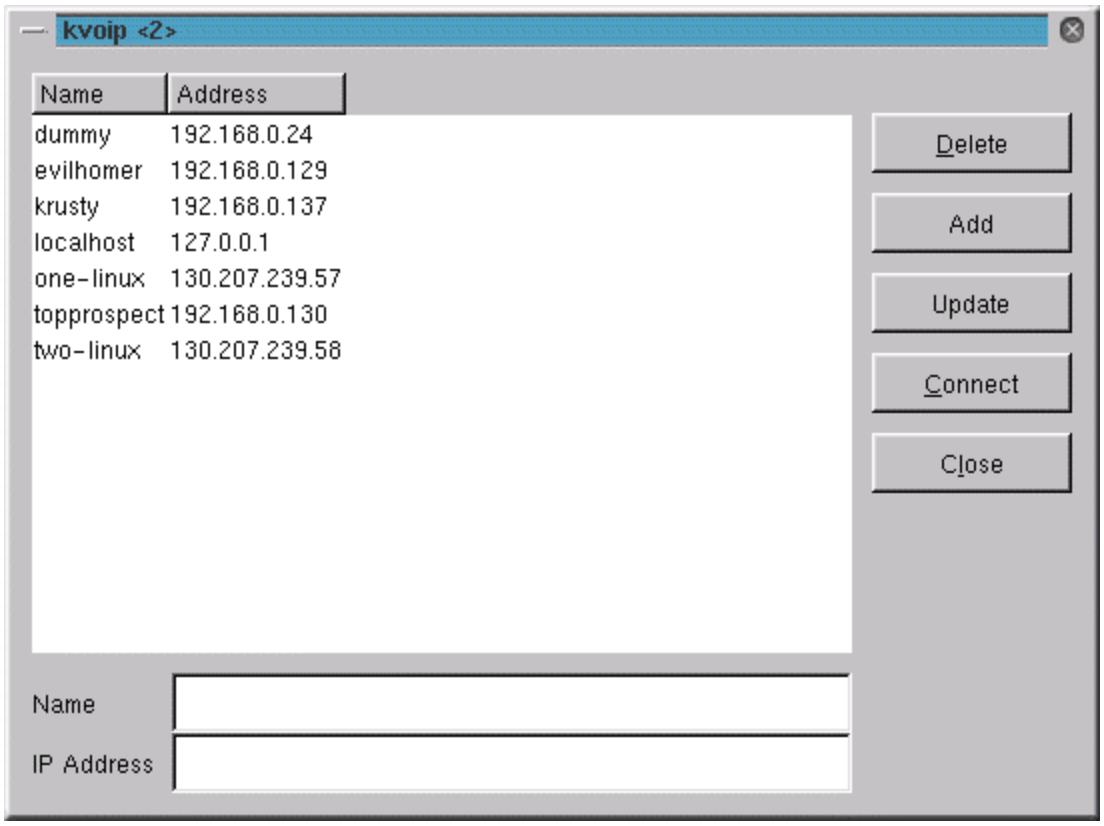
Figure 3.5 GUI Main Window Screenshot



The connection procedure is very basic. The user types in a computer DNS address or IP address into the connection box and presses connect. The user may also bring up the address book window and select an address to connect. The status bar gives the user information about what step the software is in during connection.

See the figure below for a screenshot of the address book window.

Figure 3.6 GUI Address Book Screenshot



Please see the KVoIP users manual (in an attached appendix) for more information.

4. Testing

4.1 Testing the sound and network code

Before the protocol was implemented, the basic sound and network code were first tested using a simple test program named testprog. This simple program connects to itself over the network loopback device to simulate a connection. It would copy data from the microphone, send it over then network, then receive it and play the sound through the speakers. It still functions as a basic sound tester as it plays sound but doesn't do protocol negotiation.

4.2 Testing the full backend

After protocol support was implemented, testprog was copied and extended into another program named clvoip. This simple command line driven program implements the protocol negotiation and could be set to either attempt to start a connection or wait for a connection. It was first used to shake out the bugs in the protocol while the GUI was

being integrated. It has since been used as a functional client to test KVoIP against to insure the GUI version would properly handle the protocol and connection creation.

4.3 Testing the Connection State Machine (CSM)

Before the GUI and the CSM were integrated with the backend, kvoip_stimulus was used to force the CSM through its different paths. Once the backend integration was completed, two copies of KVoIP on different machines were used to fully test the CSM. Hence, kvoip_stimulus was only implementing paths #1, #2, #3, and #4. Please see the flow chart for the "No Connection" state for a graphical representation of these paths.

4.4 Testing the full KVoIP

As mentioned before, two copies of KVoIP on different machines have been used to check the full functionality of the program. Any other bugs were found by methodically forcing KVoIP to pass through all its states in all of the allowed orders. Options were repeatedly checked to function correctly both across reconnections and upon closing and reopening KVoIP.

5. Marketing

5.1.1 Economic Feasibility

The following table displays the economic breakdown of the development of this project. Assuming that a software company undertaking a project like this one already has all the necessary hardware and software required for development. This is not an unusual assumption since the only hardware required for development is two standard networked computers. All developmental software is free and open-source.

Task	**Man-hours**	**Cost**
Backend Development	30	$900
Networking	20	$600
GUI Development	20	$600
Integration and Testing	30	$900
Maintenance and Support	120+	$3600+
		$9600+

A strenuous testing phase with various hardware/software configurations and version compatibility could be very expensive. The man-hours required setting up computers in various configurations and keeping up with installed versions and test cases would be excessive, not mention the time it would take to solve whatever problems were found.

This testing phase may be forfeited and replaced with a much longer and more frustrating maintenance and support phase. In this case, your users are much less satisfied, but they

will also do a lot of work for you. Linux users like to fix bad software that they got for free, and they often like it so much, they'll just send their fixes right back to the people who gave them the software in the first place. If user complaints and software updates can be handled appropriately, the cost of this phase may be cheaper on the whole.

5.1.2 Commercial Products

Internet telephony, the basic idea of our project, is hitting the market in a big way right now. Two popular chat programs, AOL Instant Messenger and ICQ, now include full duplex voice interaction with their latest releases. Very large, competitive companies who no doubt put a lot of research and strategic planning into their products developed these latest additions.

However, there are still some problems with these programs. Internet traffic even between two universities can sometimes be so much as to render full duplex conversations impossible. In this case, the program will default to half duplex mode. A decision is made on this issue when the conversation is initiated. Sometimes, the user will find this style of conversation so frustrating he will opt to close the voice chat and just use the text chat.

Another popular new release is Dialpad.com, which actually uses the phone system to allow an Internet user viewing their website to call someone's home phone. The real attraction for this site is in the fact that long distance calls can be made over their system for free. The site is so popular that a busy signal is expected the first four or five times you try to place a call. Unfortunately, long distance rates are getting so low that the lack of privacy and lower sound quality over this system makes it less desirable for the average user.

These products verify the economic feasibility of Internet telephony. Although the software for these products is distributed widely and freely, AOL and ICQ (now owned by AOL) intend to make money for these services through advertising. Therefore, the source code for these isproducts is protected and unavailable.

5.1.3 A Niche for KVoIP

We believe that our product could be marketable as a chat solution for Linux. While Dialpad.com is platform-independent, the most popular voice chat programs today are all developed for the Windows platform. However, a rise in interest in the Linux OS is causing many companies to consider releasing Linux versions of their software. The problem with Linux in this area is that Linux users expect your software to be open-source. The problem with releasing open-source software is that you cannot guarantee advertisers that people are seeing their ads. Easy-to-use patches would spring up quickly for removing the bothersome advertisements.

This problem aside, a company with a Windows-based chat and messaging system may be interested in buying code that implements something similar for Linux. Another case may be a company that is interested in competing with these other Internet software companies and decide that Linux is the best playing field for taking an advantage. This may very well be true. Many Linux users believe that if Linux distributors could solve their current version incompatibilities and sparse documentation problems, Linux could be a much faster Operating System than Windows for networking. If this proves true, Linux may support higher-quality and more reliable Voice-Over-IP software.

6. Conclusions

In order for KVoIP to be a commercially viable product, a few more features must be added:

- Audio compression. It cannot be assumed that the average consumer interested in a product like this will have a fast enough network connection all the way to his long-distance friends to support uncompressed audio.
- Higher quality audio. KVoIP supports up to 8-bit, 22 kHz audio data. With continued development, 16-bit, 44 kHz should be possible.
- Dynamic rate changing. A commercial KVoIP should be able to detect when the network congestion increases or decreases and adjust its audio properties accordingly.

However, in light of these short comings, KVoIP fulfills its purpose as a fully functional, voice-over-IP communicator. In addition, KVoIP implements some advanced GUI features through the magic of Qt and KDE that increase its commercial appeal considerably.

7. Documentation and Sources

7.1 Internet Sources

http://www.alsa-project.org
http://quake.sourceforge.net
http://www.kdevelop.org
http://www.troll.no/
http://developer.kde.org
http://www.gnu.org/manual/cvs/index.html
http://www.fourmilab.ch/speakfree/unix/

7.2 Book References

Stevens, W., “Unix Network Programming Vol. 1”

8. Appendices

8.1 Source Code

APPENDIX A.

KVOIP SOURCE CODE

8.2 User's Manual

APPENDIX B.

KVOIP USER'S MANUAL

8.3 Presentation Slides

APPENDIX C.

KVOIP PRESENTATION SLIDES