Lienhuachih Acoustic Monitoring System

Allison Johnston

August 30, 2010

University of California- San Diego

Taiwan Forestry Research Institute

Abstract:

The Taiwan Forestry Research Institute (TFRI) has an extensive wireless sensor network at their Lienhuachih field site. This report describes the integration of an acoustic monitoring system for frog calls into the existing sensor network. The acoustic monitoring system uses data turbine, an open source streaming data management program to provide access to data in real time. Data turbine uses a Ring Buffer Network Bus (RBNB) server to temporarily store data that may be accessed by multiple data processing and storage applications. The RBNB server can accommodate many heterogeneous data streams which may be viewed simultaneously in real time with Real Data Viewer (RDV).

Intro:

The Taiwan Forestry Research Institute (TFRI) has nine field sites in Taiwan, all with many sensors for gathering data for ecological studies. Many of these sensors are remote and sometimes inaccessible due to typhoon damage. This necessitates a remote connection to the sensors so that they may be efficiently maintained. The traditional method of using a data logger to collect data for a set amount of time and then go to the field to download data from the logger is inefficient in these remote to inaccessible areas. For this reason, TFRI has dedicated many resources to building wireless sensor networks. This effort has made TFRI one of the leading ecological research institutes in sensor networking and eco informatics.

TFRI uses data turbine, an open source streaming data middleware program that was developed by researchers at Creare and the San Diego Supercomputer Center (SDSC) and funded by the National Science Foundation (NSF). [1] Data turbine is used to consolidate TFRI's many streams of real time data from its sensors in remote field sites. Data turbine consists of a ring buffer network bus server (RBNB) that stores data in a temporary buffer.[1] As of June 2010, TFRI's Lienhuachih field site had meteorological and video data connected to a RBNB server at the field site.

However, audio data from a field microphone was not connected to the RBNB server and was being sampled using TotalRecorder, an audio ripping program, and stored in an FTP server in Lienhuachih. This was inconvenient because the audio data was not available in real time and it was hard to view audio and meteorological data simultaneously. Therefore, the goal of this project was to connect audio data to the main Lienhuachih RBNB server, store samples of the audio data for future research, and facilitate processing of the audio data. The proposed network architecture for accomplishing that goal is shown in figure 1.

Figure 1. Network architecture for the acoustic monitoring system.

The field microphone stationed on the bank of a pond for capturing frog calls. The set up is shown in figure 2. The pond contains three species of frogs, one of which is the rare harp frog. The harp frog is only known to live in Taiwan so it is an important object of study. In the next year, TFRI researchers would like to use the acoustic data from the frog pond to identify which characteristics of frog mating calls are most successful.[8] Further in the future, TFRI's goal is to use an acoustic monitoring network to identify which frog species are present at any given time.[8]

Figure 2. Lienhuachih Frog Pond: Three species of frog live in this pond. The field microphone is on the tripod on the left and the field computer is in the box on the right.

Theory:

Data turbine systems include three components, sources, sinks, and at least one RBNB server. Source programs connect the streaming output of a sensor or data logger to the RBNB server. The data enters the ring buffer where it is organized by the time it entered the ring. As more data fills the ring, the oldest data is discarded to make room for the newest data. The size of the ring is determined in the source program. While the data is in the buffer, it is accessible sink programs. Sink programs connect the data in RBNB to data processing applications or databases. Data turbine has a java library that can be used to write these programs. [1,]
[4]

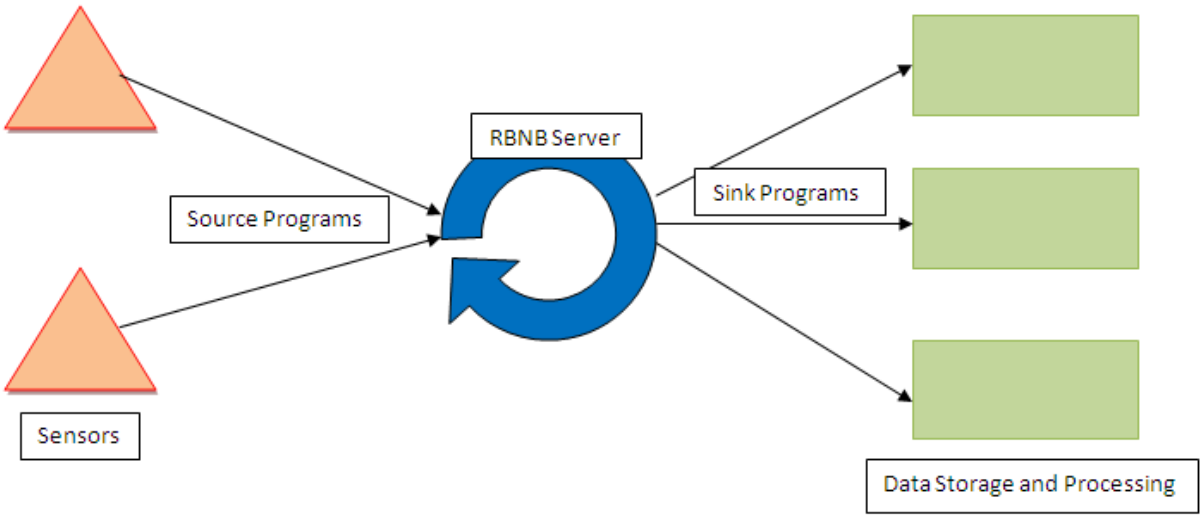


Figure 3 Basic data turbine system: Sensors are connected to an RBNB server through Java source programs. Data storage and processing applications access the sensor data through Java sink programs. There may be any number of sources and sinks.

Procedure:

In order to implement the system in figure 1, three major connections needed to be made. First, the microphone had to be connected to the RBNB server on the field computer. This was done by modifying an existing java program written by Michael Nekrasov, Generic Sound Source. Generic Sound Source takes the microphone line in and converts it to a byte array. The byte array can then be sent to the RBNB server using the data turbine Java library. This program determines the format of the audio signal, so the code must be changed if the audio format changes. Table 1 shows the audio format variables and their default values.

Table 1. Audio format variables and their defaults: The Generic Sound Source program contains these default audio format values. Their values must be changed in the code if the audio format does not match these defaults.

Audio Format Variable	Default
Sampling Rate	8,000 Hz
Sampling Time	5 s
Sample Size	16 bits
Number of Channels	1

Big Endian	False
Signed	True

After the audio data is sent to the RBNB server on the field PC, it must reach the RBNB server in the Lienhuachih server room. This was achieved using the 'mirror' tool in the data turbine Admin feature. The mirror allows a source on one RBNB server to be streamed to another RBNB server.[7] In this instance, the AUDIO source in the field computer's RBNB server was mirrored to the server room computer. This was done because the server room RBNB has a global IP address so it is directly accessible outside of the Lienhuachih network. Also, sending the audio source to the main RBNB server makes the audio data available in the same server as the rest of the data collected at Lienhuachih. Detailed instructions for mirroring RBNB servers were created for future use at TFRI. The document is included on page seven in the appendix.

The audio data in the Lienhuachih RBNB server is accessible to researchers and the public using the Remote Data Viewer (RDV). RDV plays real time audio data, shows graphs of streaming numeric data, and plays video. This a good tool for streaming data, but it can only show data in the RBNB server's archive and does not allow researchers to store data. Therefore, a permanent storage option was necessary.

In order to permanently store the audio data collected by the acoustic monitoring network, the audio data must be converted to a universal audio file format and sent to permanent storage. This was achieved by writing a Java program called RBNB to FTP. The program acts as an RBNB sink and takes audio data in the form of a byte array stream and writes it to a wave file, TFRI's preferred audio format. The program then sends the file to an FTP server using the Java FTP client library. There are many command line options so that the program is general enough to apply to other acoustic networks without too much modification. The options include the RBNB server IP and port, the FTP server IP and port, the destination directory for the audio files in the FTP server, and the recording interval.

The final result of the acoustic monitoring network is the analysis of the data that is collected. Though this project mainly focused on implementing the computer network for the acoustic monitoring system, I wanted to help the researchers find the right tools for analysis. TFRI does not own the rights to MATLAB, the natural first choice for signal processing. Scilab was the best open source alternative to MATLAB since it has nearly all of its signal processing algorithms, has the ability to translate MATLAB m-files, and has an open source equivalent to SIMULINK called Xcos.[5] "How to Use Scilab Audio Processing Tools" on page eleven in the appendix is a reference created to explain basic signal processing in Scilab for the TFRI researchers to get started.

Results:

The acoustic monitoring system ran smoothly for three days in between tinkering and testing. All of the pieces are in place to deploy the system full time, but the remote connection to the virtual network server on the field computer is down due to a thunderstorm. This means that the network cannot be deployed until the virtual network server is fixed. I have taught one of the TFRI researchers, Dr. Wang, everything he needs to know to do this so when the server is fixed he will be able to deploy the system.

Discussion and Conclusion:

The acoustic monitoring network is very useful in that it has the capability to play the frog pond sounds in real time, and it provides permanent storage for the acoustic data. This way, researchers will have the capability to set up real time signal processing applications and be able to simultaneously hear the Lienhuachih pond audio signal and see meteorological data or any other kind of data in RBNB. The permanent storage aspect of the system facilitates long term studies of ecological trends.

There is, of course, much more work to be done for the acoustic monitoring system a few projects that I recommend for making the system better are as follows:

- Write a batch file to run every time the field computer is rebooted so that even when the computer restarts without warning in a thunderstorm, the field computer will run RBNB and Generic Sound Source.

- Set up real time signal processing using MATLAB and Simulink. It is likely that the audio data will need filtering since there is a lot of background noise, and this may be done in real time by creating a Simulink model.

- Improve the RBNB to FTP program.

    - This program sometimes crashes unexpectedly, so with better error handling it will be more robust.

    - Find a way to send the file to the FTP server directly, without saving it to the working directory first.

- Replace the field computer with a smaller single board computer, such as a Gumstix. These computers use a fraction of the power of a traditional computer and cost a fraction of the amount.

Appendix:

References:

1. "Overview." *Open Source Data Turbine Initiative.* 12 July 2010. <http://dataturbine.org/content/overview>.

2. "Product Support." *Gumstix- Linux computers the size of a stick of gum.* 15 August 2010. <gumstix.com>.

3. *Taiwan Forestry Research Institute*. 27 July 2010. <tfri.gov.tw>.

4. Nekrasov, Michael. Personal Interview. CalIT^2. Wednesdays, March 2010 through June 2010.

5. "Scilab Support." *Scilab: The Free Platform for Numerical Computation.* 24 August 2010.

<http://www.scilab.org/support>

6. “Standard Edition 6 API Specification.” *Java™ Platform*. 25 July 2010. <http://java.sun.com/javase/6/docs/api/>.

7. “Getting Started with Data Turbine” *Network for Earthquake Engineering Simulation.* 12 July 2010. <https://www.nees.org/research/dl_detail/getting_started_with_data_turbine_31/#10>.

8. Chun-Wen. Personal Interview. Lienhuachih. 15 July 2010, 14 August 2010.

Mirroring Documentation:

Mirroring RBNB Servers:

Mirroring RBNB servers is a method of streaming data from one RBNB server to another. It allows the user to add a source to her local RBNB server from a global RBNB server or send a source from her local server to a global target RBNB server.

This allows users to aggregate data streams from disparate servers, increasing the ease of live data comparison and processing. It is also useful for communicating streams of real time data without losing functionality, since a mirrored data source may be used in any way that a local data source may be used.

How to Mirror From RBNB Servers:

Mirroring *from* an RBNB server means taking a data source from a global RBNB server and putting it into a local RBNB server. This is also called *pull* based mirroring. In the following instructions, the local RBNB server is the server that will be receiving the desired data source.

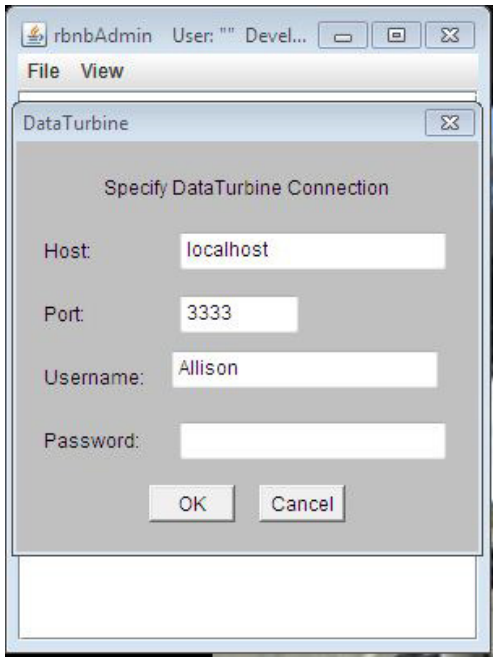
1. Execute admin.jar, found in the bin directory of the RBNB home directory.

2. Go to File => Open

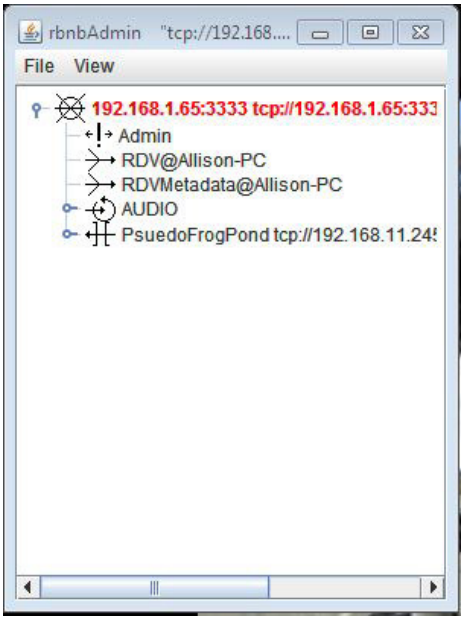
3. The menu "Specify DataTurbine Connection" will pop up.

    a. Enter the host and port of the local RBNB server.

b. The username field must contain at least one character. Unless the local server is password protected, the username is arbitrary.

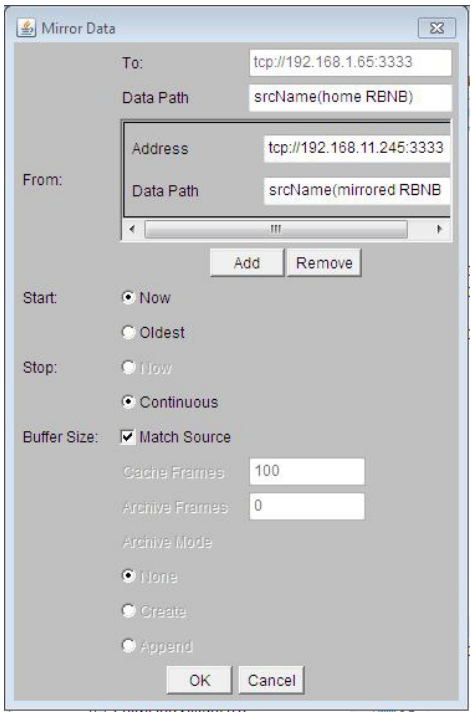


4. After about a few moments, the connection you specified should show up in red text.



5. Right click on the red text and choose “Mirror From.”

6. In the “Mirror Data” menu, “To:” is your local RBNB server address. “Data Path” will be the *source name* of the channels you are mirroring in your local server. The “From: Address” field refers to the address of the RBNB server containing the desired channel. It is in the form tcp://server’s IP:port. “From: Data Path” is the name of the source you wish to mirror. Note that you chose the source and then all the channels within that source are mirrored.



7. If the mirror is successful, you should see the To: Data Path you specified in the RBNB Admin screen.

8. Troubleshooting:

    a. If the desired data source is on an RBNB server outside of the local network, make sure to use the public IP address of the RBNB server.

    b. The “From: Data Path” must be identical to the name of the desired data *source*, not the channel name! Remember: in RBNB terms sources are folders of channels.

    c. Be patient! Sometimes it takes a few minutes to find the mirrored channel’s sources.

How to Mirror To RBNB Servers:

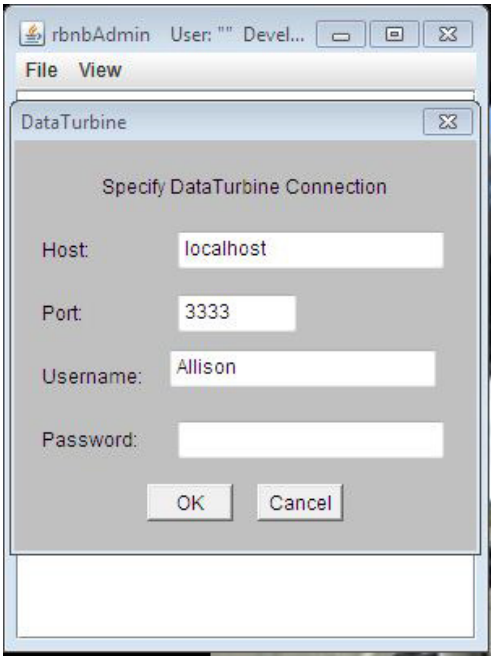
Mirroring to an RBNB server means sending a data source from a local RBNB server to a target RBNB server. It is an example of *push* based architecture.

1. Start the admin.jar file in the bin file of the RBNB home directory.

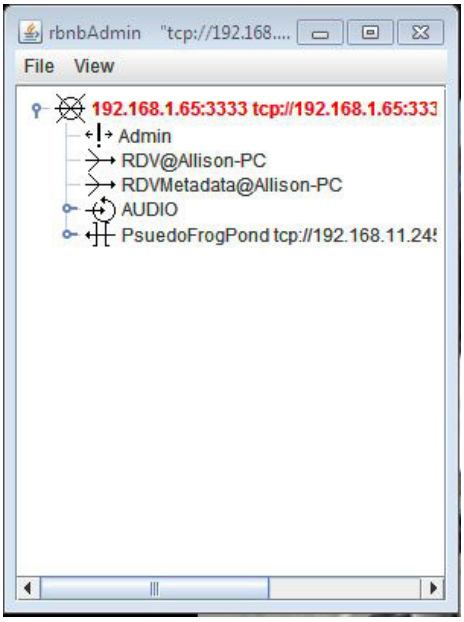
2. Go to File > Open

3. The menu "Specify DataTurbine Connection" should pop up.

    a. Username must be specified- if you don't have one anything will do so long as the server is not password protected.

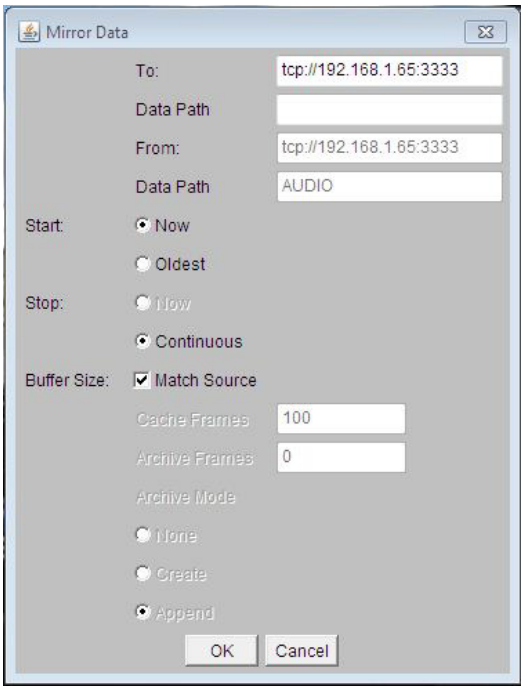


4. After about a minute, the connection you specified should show up in red text.



5. Right click on the channel you would like to send to the non-local RBNB server and choose the "Mirror To" option.

6. In the "Mirror Data" menu, "To:" is the address of the target RBNB server. "Data Path" is the name of the source when it gets to the target server. "From:" is the address of the local RBNB server. It should already be filled in. The second "Data Path" is the name of the source that is sent to the target server. It should also already be filled in.



7. The connection is successful if "_Mirror.TargetDataPath TargetServerIP" appears in the Admin Screen. This may take a few minutes or require refreshing.

Scilab Audio Signal Processing Documentation:

How to Use Scilab Audio Processing Tools:

Overview:

Scilab is its own programming language that is designed to be used for many scientific computations. It is made to be user-friendly and has many useful prewritten functions for audio processing. The functions take the form:

[output1, output2] = function(parameter1, parameter2)

The output variables to the function may be given any name desired. If no output variable name is specified, then its value is simply displayed in the console and cannot be used later. Type *browsevar* in the console to see a list of the current variables.

The Scilab console allows you to enter a single command and get an immediate output. You may also write your own functions using the Scilab text editor. This will be helpful for automating a series of commands that you use over and over.

Some useful hints:

- Use a semicolon (;) after a command to stop the output from displaying on the screen. Often vectors will be thousands of entries long, which can be tedious to display on the screen. The semicolon will prevent the command's output from being displayed, but it will not affect the command's action in any other way.

- The colon (:) symbolizes a for loop. The command *y = 0:10;* means that y is a vector with the numbers zero through 10. The command *y = 0:0.5:10;* means that y is the vector <0, 0.5, 1.0, …, 10>. The middle number is the step interval.

- Always use the asterisk (*) for multiplication. 10x is a variable name, while 10*x is ten times the variable x.

- Typing *help function_name* displays the documentation for that command in the help browser.

- Scilab and Matlab are very similar, but Matlab has better documentation. Often, it will be easier to look up a function in Matlab and then try doing the same thing in Scilab. Also, this website gives the Scilab equivalent of many Matlab commands: http://www.scilab.org/product/dic-mat-sci/M2SCI_doc.htm

- The % signifies a comment, written to explain what is going on in the command, but not a part of the command.

Plotting:

plot(x, y)

The plot() command is used to make graphs in Scilab. The above command plots vector x versus vector y. The two vectors must be the same length.

plot2d(y)

The plot2d command plots the vector's entries versus their position in the vector, i.e. versus the default vector <1, 2, 3, …, n>, where *n = size(y, "*")*. If there is not already a plot open, a new graphical window will open. If not, Scilab will plot the function on an existing graphical window. You can change the

graph's axis limits, labels, tick marks, and scale using the Axes Properties window (Edit > Axes Properties).

You may save the graph using file > save, or copy it to the clipboard (Edit > Copy to Clipboard) so that other applications can access the graph.

More Info: http://www.scilab.org/product/man/plot.html

Loading an Audio File:

[signal, sampling_rate, bit] = wavread("pathname");

To load a .wav audio file, use the wavread command like above. The pathname is the location of the file in your computer's file system. Make sure to put the pathname in quotes.

The signal output will be a vector with the magnitude at each sample of the audio signal. The sampling rate is the rate at which the audio signal was sampled, in hertz. Bit is the number of bits in each sample. The signal vector will be the most important output of this function. The sampling rate and number of bits are simply information on the audio format of the file.

More info: http://www.scilab.org/product/man/wavread.html

auread also works in the same way as wavread, but for .au audio files. More info: http://laris.fesb.hr/digitalno_vodjenje/download/scilab/sound/auread.htm

Graphing Amplitude vs. Time:

plot2d(signal)

This command will plot the signal vector from the wavread command versus the default x vector. This can give you a quick idea of the shape of the signal, but the x-axis will be the sample number of the signal, not time. In order to plot amplitude vs. time, use the following command:

n = size(signal, "");*

t = (0:n-1)/fs;

plot(t, signal)

where t is the time range of the signal, n is the number of samples of the signal, and fs is the sampling

rate of the signal. This will give you the same graph, but with more useful axis numbering in seconds. Below is an example script for producing a graph of a signal's time versus magnitude.

```
-->[x, fs, bits] = wavread("C:\Users\Allison\Downloads\frog.wav");          % Load the
wave file frog.wav

-->t = (0:n-1)/fs;                                                         % Make a time
range vector

-->plot(t, x);                                                             % Plot time
                                                                           versus
                                                                           magnitude. %
                                                                           The graph is
                                                                           shown in figure
                                                                           % 1.
```

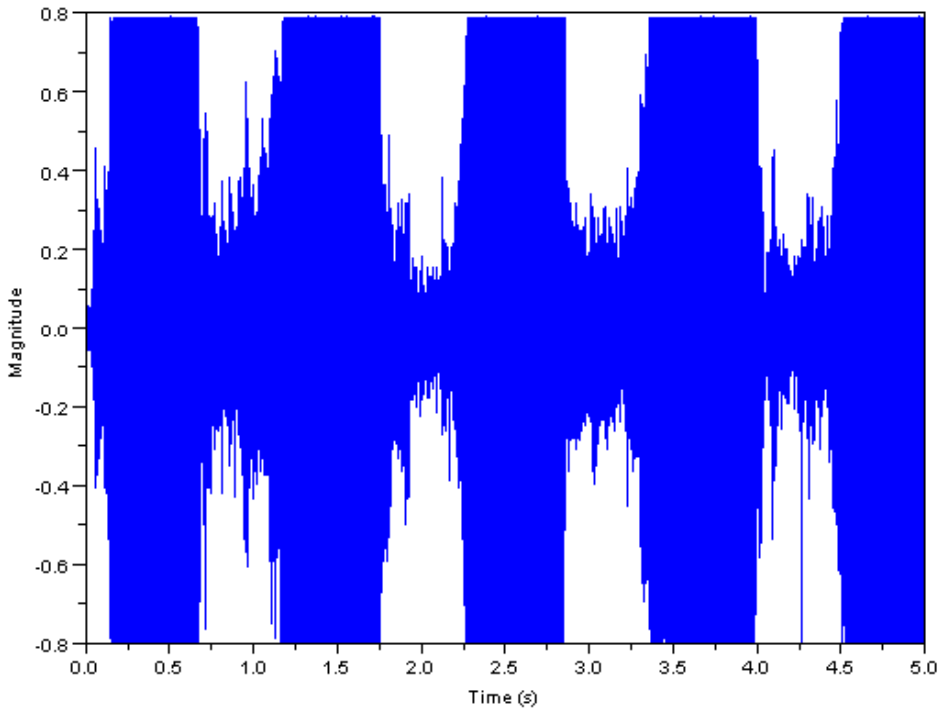


Figure 4: Time vs. magnitude of a bull frog call.

Fourier Transform:

y = fft(x);

The above command takes a signal x and computes its discrete fourier transform. In order to graph the frequency versus the power of the signal, use a variation of the following script:

-->[id, filename] = diary("C:\Users\Allison\Documents\PRIME", "new") % Turns on the “diary” and begins saving the % console session into a text file in the location % specified.

-->[x, fs, bits] = wavread("C:\Users\Allison\Downloads\frog.wav"); % Loads the file frog.wav. Each audio sample % is stored in the vector x, fs is

the

% sampling frequency of the

signal (in Hz)

% and bits is the number of bits

in each sample

-->n = size(x, "*"); % n is the number of samples of
the signal,

% or the size of the vector x.

-->t = (0:n-1)/fs; % t is the time range of the
samples.

-->freq_range0 = (-n/2:n/2-1)*(fs/n); % freq_range0 is the symmetric
frequency

% range of the signal. Keep in
mind that if the

% frequency of the signal is

higher than

% fs/2, the Nyquist frequency,

aliasing will

% occur and the signal may be

misrepresented.

-->y = fft(x); % y is the Fourier transform of
the signal.

% It represents the magnitude of the signal

% in the frequency domain.

-->power = (abs(y).^2)/n; % power is the power of the signal in the

%frequency domain.

-->power0 = fftshift(power); % power0 is the power of the signal shifted

% so that it is centered on 0.

-->plot(freq_range0, power0) % this plots the frequency versus power, shown % in figure 2.

-->ydb = 10*log10(power); % ydb is the power of the signal in decibels

-->plot(freq_range0, ydb) % this plots the frequency versus power in % decibels.

-->freq_abs = abs(freq_range0);

```
-->plot(freq_abs, power0)                % this plots the frequency
versus power in the
                                         % positive quartile only


-->plot(freq_range0, abs(ydb))           % this plots the absolute value
of the signal's
                                         % power in decibels, shown in
figure 3.


-->diary(0)                              % exits out of diary mode, no
longer writing the
                                         % console entries to a text file.
```

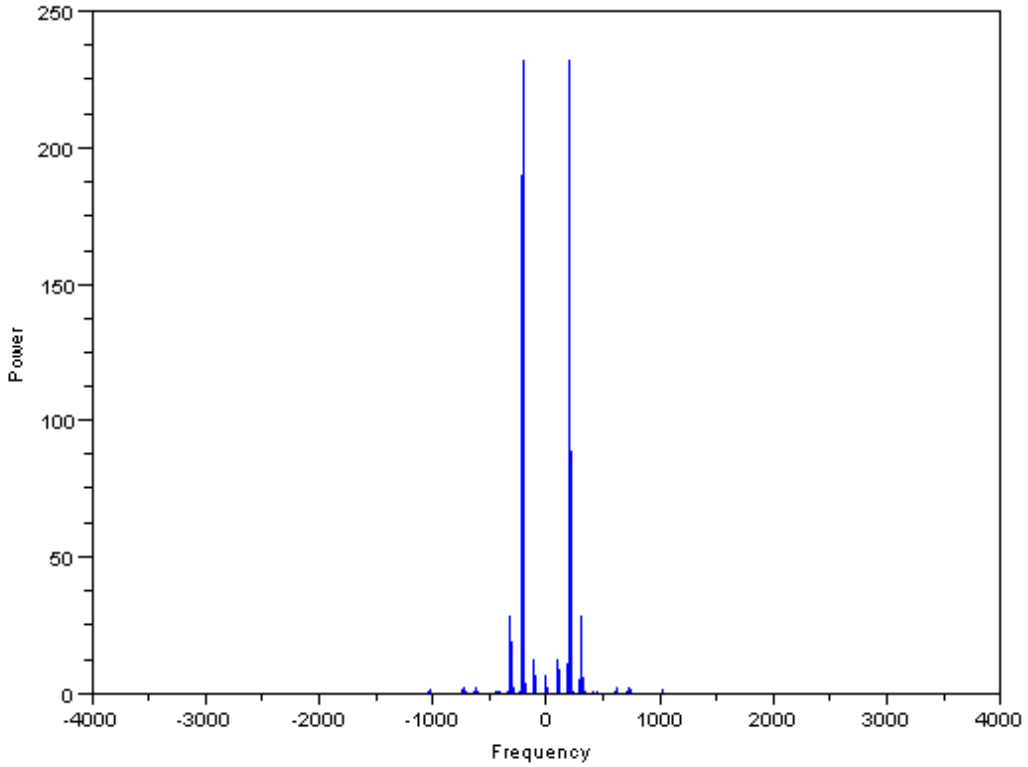
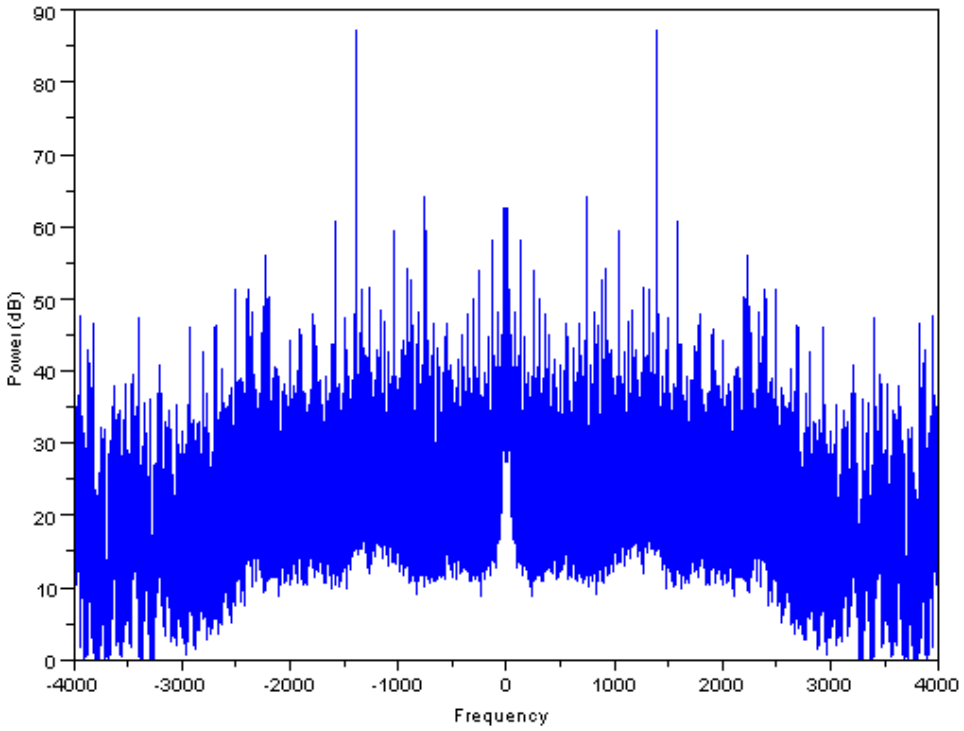


Figure 5 Frequency versus power of a bull frog call.

In order to graph the frequency versus power in decibels, use the formula $y_{db} = 10\ log_{10}(power) = 20 log_{10}(y_{mag})$.

Figure 6 Frequency vs. power in decibels of a bull frog call.



This table from the Matlab help utility contains some useful commands:

Quantity	Description
`x`	Sampled data
`m = length(x)`	Window length (number of samples)
`fs`	Samples/unit time
`dt = 1/fs`	Time increment per sample
`t = (0:m-1)/fs`	Time range for data
`y = fft(x,n)`	Discrete Fourier transform (DFT)
`abs(y)`	Amplitude of the DFT
`(abs(y).^2)/n`	Power of the DFT
`fs/n`	Frequency increment
`f = (0:n-1)*(fs/n)`	Frequency range
`fs/2`	Nyquist frequency