ABSTRACT

Nowadays, the most popular media for transmitting information are metal cables, electromagnetic waves in the air, as well as fiber optics. However, it is possible to use other signals too – for example, sound. Audio at 23 kilohertz is outside the range of human hearing but surprisingly can be produced/picked up by a cheap laptop speakers/microphone. The main objective of the project is to reproduce the work described in a blog post and present a demo transmission using GNU Radio.

INTRODUCTION

The aim of this report is to analyze the blog post by a developer nicknamed *anfractuosity* and also give an end-to-end manual for running the actual application. This particular blog post describes a way to pass data over the air using ultrasound.

Ultrasound is defined as all the wavelengths higher than the upper audible region of human hearing, usually around 20 kilohertz. A typical computer microphone can, however, receive higher frequencies and basically all the speakers can produce much higher frequencies. This is perfect for our use case. We would like to transfer data using as simple hardware as possible without interfering with human hearing in any way.

The blog post features a video of the proof-of-concept application running on two separate laptops. No audible sound is received, yet the data is transferred from one machine to another.

USED SOFTWARE

- GNU Radio\(^3\) is a free software development toolkit that provides signal processing blocks to implement software-defined radios and signal-processing systems. It also includes a graphical UI for creating a flowgraph of each step in processing the data. Such flowgraphs are “compiled” into a executable Python scripts.
- Netcat\(^4\) is a computer networking utility for reading from and writing to network connections using TCP or UDP. This experiment uses *netcat* for sending data over TCP to the running Python application, the same applies for receiving data.

DESIGN

The project consists of two applications, one of which is converting the text into sound waves and other which receives the soundwaves through microphone (sender and receiver applications). Both the sender and receiver application are composed of different modules that are provided by GNU Radio and modify the data in a certain way.

---

\(^2\) [https://github.com/anfractuosity](https://github.com/anfractuosity)  
Both applications are described in GNU Radio as shown in the following flowgraphs:

- **Sender.grc**

  ![Flowgraph for Sender.grc](image)

  - TCP Source
    - Address: 127.0.0.1
    - Port: 10.004k
    - Modes: Server
  - GFSK Mod
    - Samples/Symbol: 9
    - Sensitivity: 1
    - BT: 350m
  - Packet Encoder
    - Samples/Symbol: 9
    - Bks/Symbol: 1
    - Preamble
    - Access Code: Pad for USRP
    - No Payload Length: 5
  - Rational Resampler
    - Interpolation: 500
    - Decimation: 1
    - Taps: 0k
    - Fractional BW: 0
  - Frequency Xlating FIR Filter
    - Decimation: 1
    - Taps: fir.filters.low_pass
    - Center Frequency: -28k
    - Sample Rate: 48k
  - Complex To Real

- **Receiver.grc**

  ![Flowgraph for Receiver.grc](image)

  - Audio Source
    - Sample Rate: 48KHz
  - Float To Complex
  - Frequency Xlating FIR Filter
    - Decimation: 1
    - Taps: fir.filters.low_pass
    - Center Frequency: 28k
    - Sample Rate: 48k
  - Rational Resampler
    - Interpolation: 1
    - Decimation: 500
    - Taps: 0k
    - Fractional BW: 0
  - GFSK Demod
    - Samples/Symbol: 9
    - Sensitivity: 1
    - Gain Mu: 175m
    - Mu: 500m
    - Omega Relative Limit: 5m
    - Freq Error: 0
  - TCP Sink
    - Address: 127.0.0.1
    - Port: 10.006k
    - Modes: Server

The modules are pretty much the same for both applications acting as mirrors of each other.

- **TCP Source** - acts as an input for the entire sender application, waits for a TCP connection to be created, accepts it and directs all incoming data to the next
TCP Sink acts as the output for the receiver application, similarly to the TCP Source, it waits for a connection, however instead of receiving data, it sends all the incoming data (from the connected module) to the connecting party.

Audio Source is used by the receiver application, uses the host machine’s input device (microphone) and passes the incoming data to the next module.

Audio Sink passes the incoming data to host machine’s output device (speakers), used by sender application.

Packet Encoder wraps the incoming bytes into a packet of a given payload length with a header, access code and preamble.

Packet Decoder reads the header of the packet to get the payload length, unwraps the packet and outputs the payload.

GFSK Modulator (Gaussian frequency-shift keying modulator) block which reads in binary data and modulates the carrier frequency with the input data. Outputs the modulated frequency.

GFSK Demodulator reads in the modulated frequency and demodulates it back into binary data, also outputs the binary data.

Rational Resampler used to converts from one sample rate to another using the following ratio: output frequency = input frequency * (interpolation / decimation).

Frequency Xlating FIR Filter implements a frequency-translating Finite Impulse Response filter. It performs frequency translation, channel selection and decimation in one step. Input and output are both complex signals.

The sender application will initially wait until a successful TCP connection is created. After that all the received messages will be wrapped into well-formed packets, then modulated into complex signal, sampled at 48 kHz (standard in professional audio equipment), converted from complex signal to real values and sent to the audio output device (speakers).

Receiver does the same listed steps, only in reverse order. Also the application won’t start before a successful TCP connection is created. This time, however, microphone acts as the input device. After all the conversion steps, the TCP Sink module will send the data to the specified IP address.

**TESTING**

Both the receiver and sender applications were running on a single machine - MacBook Pro 15 inches. Using the same values as the original creator yields very reliable results,
external noise doesn’t seem to be interfering with the data transfer at all. Also, the carrier frequency needs to be around 23 kHz. Higher frequencies aren’t picked up by cheap microphones. Lower frequencies on the other hand will be audible to human ears and are not suitable for real use cases.

By adjusting different parameters, I was trying to model a situation where the receiver would receive incorrect data. But either the data wasn’t received at all or it was always correct. This is likely due to wrapping the data into packets so the checksum must match or nothing is received. Also, the most reliable data transfer seemed to be happening between a laptop’s internal microphone and speakers instead of using external ones, however, it’s related to their physical closeness and little external noise.

**ENCOUNTERED TECHNICAL PROBLEMS**

All the issues I had, were related to setting up GNU Radio. What made it complicated for me was that I’m running Mac OS instead of a Linux distribution.

- **First try - using a VirtualBox Ubuntu image**
  - Initially I thought of installing GNU Radio on a Ubuntu virtual machine. It seemed to go well but for some reason I couldn’t pass the input/output devices properly from host machine to virtual machine so the sounds coming to microphone weren’t picked up.

- **Second try - GNU Radio custom version for Mac**
  - After a while I succeeded in installing GNU Radio properly but there seemed to be some libraries missing so I couldn’t use any modules which used WX GUI.

- **Third try - Installing GNU Radio with MacPorts**
  - I had the same issue as with the previous installation method.

- **Fourth try - Building GNU Radio from source with all the necessary libraries**
  - After some experimenting I managed to also include the missing WX GUI libraries to installation process and build everything from source, now the application seemed to run as expected.

5 [https://github.com/cfriedt/gnuradio-for-mac-without-macports](https://github.com/cfriedt/gnuradio-for-mac-without-macports)

6 [https://www.macports.org/](https://www.macports.org/)
SIMILAR WORKS

I also searched for information on some other similar applications. It turns out there are several working applications which achieve similar results:

- **QuietJS**: This is a javascript binding for libquiet, a library for sending and receiving data via sound card. It can function either via speaker or cable (e.g., 3.5mm). The webpage contains a working application which can send and receive data similarly to the solution described in this report.

- **Minimodem**: Minimodem is a command-line program which decodes (or generates) audio modem tones at any specified baud rate, using various framing protocols. It can be used to transfer data between nearby computers using an audio cable (or just via sound waves), or between remote computers using radio, telephone, or another audio communications medium.

- **Quietnet**: Simple chat program using near ultrasonic frequencies. It is written in Python and depends on numpy and pyaudio libraries.

END-TO-END GUIDE

This guide focuses mainly on Mac OS.

- **Prerequisites for GNU Radio**
  - XQuartz: download the .dmg from the webpage and start the auto-installer
  - Python - should be installed by default
  - Git - optional, but simplifies the download process

- **Installing GNU Radio**
  - My recommendation would be to use the build script provided by `cfriedt`. Clone the Git repository and run `build.sh`.

---

7 [https://quiet.github.io/quiet-js/](https://quiet.github.io/quiet-js/)
9 [https://github.com/Katee/quietnet](https://github.com/Katee/quietnet)
10 [https://www.xquartz.org/](https://www.xquartz.org/)
Cloning into 'gnuradio-for-mac-without-macports'...
remote: Counting objects: 534, done.
remote: Total 534 (delta 0), reused 0 (delta 0), pack-reused 534
Receiving objects: 100% (534/534), 675.55 KiB | 268.00 KiB/s, done.
Resolving deltas: 100% (318/318), done.

cd gnuradio-for-mac-without-macports

gnuradio-for-mac-without-macports git/master

sh build.sh

...

* Running GNU Radio
  ● Since GNU Radio also bundles a graphical UI, it needs to have a `DISPLAY` variable set in the environment. The easiest way would be to start previously installed XQuartz/X11 and launch a terminal through it. This terminal will automatically set all the necessary environment variables, thus execute:

```
❯ gnuradio-companion
```

* Running sender/receiver applications
  ● Open both applications by clicking to File > Open and navigating to `receive/send.grc` files.
  ● Run both applications by either clicking F6 or clicking the white arrow on the toolbar. It is expected that nothing happens, both applications stay listening for a successful TCP connection.

* Connecting to sender and receiver
  ● For this task I recommend to use netcat which is installed into UNIX based systems by default. The receiver is listening on port 10005 and the sender on 10004.

```
❯ nc 127.0.0.1 10005
❯ nc 127.0.0.1 10004
```

In case of successful connections, both applications now display a graph of incoming wavelengths (WX GUI FFT Sink).
In the sender Netcat connection, the data will be sent after pressing the return key. Note
that the data will be sent in blocks of 5 characters so a shorter message will not be sent before padding it with extra data (for example newline characters).

Sender
❯ nc 127.0.0.1 10004
Hello

Receiver
❯ nc 127.0.0.1 10005
Hello

CONCLUSION

I could see multiple theoretical use cases where such a simple way to share data would come in handy. However I also feel that in practically every such case there is a better solution already existing, mainly through electromagnetic waves. Also the speed is a limiting factor, since the frequency is not nearly as high as “traditional” protocols use, we can never reach such speeds over ultrasound.

There is also a issue of the ethical factor. Although most humans can’t hear frequencies over ~21kHz, there are still very many living organisms who operate on those frequencies and everday use of such applications would be highly disruptive.

All in all the experiment has been an interesting experience by both giving me technical details about the used tools and also theoretical knowledge about sound waves and their different modifications.