HOMEGROWN ACOUSTIC DATA TRANSMISSION AN EXAMINATION OF WAVE-GUI

Sam Ellington, Valentin Titenko, Dexter Chan, Mira Yun, Leonidas Deligiannidis

School of Computing and Data Science
Wentworth Institute of Technology (UNITED STATES)

Abstract

Undergraduate courses and laboratories on wireless networks and technologies have been developed in response to increasing demand for wireless networking skills in almost all industries. Since most of wireless technologies and applications require greater research and are heavily based on theoretical concepts, developing hands-on activities for undergraduates is crucial to improve the performance and engagements in the classroom. Acoustic data transmission (ADT) functions by transmitting data over the air in the form of sound with the use of a speaker and receiving data with a microphone. In this paper, we introduce ADT technology into the undergraduate classroom by using an ADT application called Wave-GUI. This paper presents a step-by-step procedure for implementing and evaluating an ADT application with only two computers. Through the testing phase, we found that ADT typically works best with high-bitrate protocols, and that additional external 'noise' does not have as much of an impact on this form of communication as one might expect. Additionally, we determined that the two main factors in the success of an acoustic data transmission were the quality of the hardware being used and microphone-speaker proximity.

Keywords: Acoustic Data Transmission, Wireless Hands-on Project, Wave-GUI Testing.

1 INTRODUCTION

From smartphones to industrial equipment, hundreds of thousands of smart devices require different forms of wireless connectivity [1]. In response to increasing demand for wireless networking skills in almost all industries, undergraduate courses and laboratories on wireless networks and technologies have recently been developed and deployed. Wireless technologies are relatively new. Developing hands-on activities covering advanced topics for undergraduates is crucial to improve their performance and engagement.

Many wireless technologies and applications require extensive research to understand and are based heavily on theoretical concepts. In order to deliver theoretical concepts and implementations into the undergraduate classroom, WiFi, Bluetooth, near-field communication, LiFi, or visible light communication, Global Positioning System (GPS), and Cellular Long Term Evolution (LTE) have been used to develop various hands-on activities and laboratories. As a new field, we introduce acoustic data transmission (ADT) technology into the undergraduate classroom. ADT functions by transmitting data over the air in the form of soundwaves with the use of a transmitting speaker and a receiving microphone [2]. To better understand the capabilities of this technology, we focus our attention on an application called Wave-GUI [3]. We chose this application because it was the best non-commercial ADT implementation we could find. We chose a free application because we believe that proprietary ADT implementations would not be emblematic of ADT on the whole. By testing its capabilities, we can better appreciate the potential of ADT, and find applications of this mode of communication for our everyday needs.

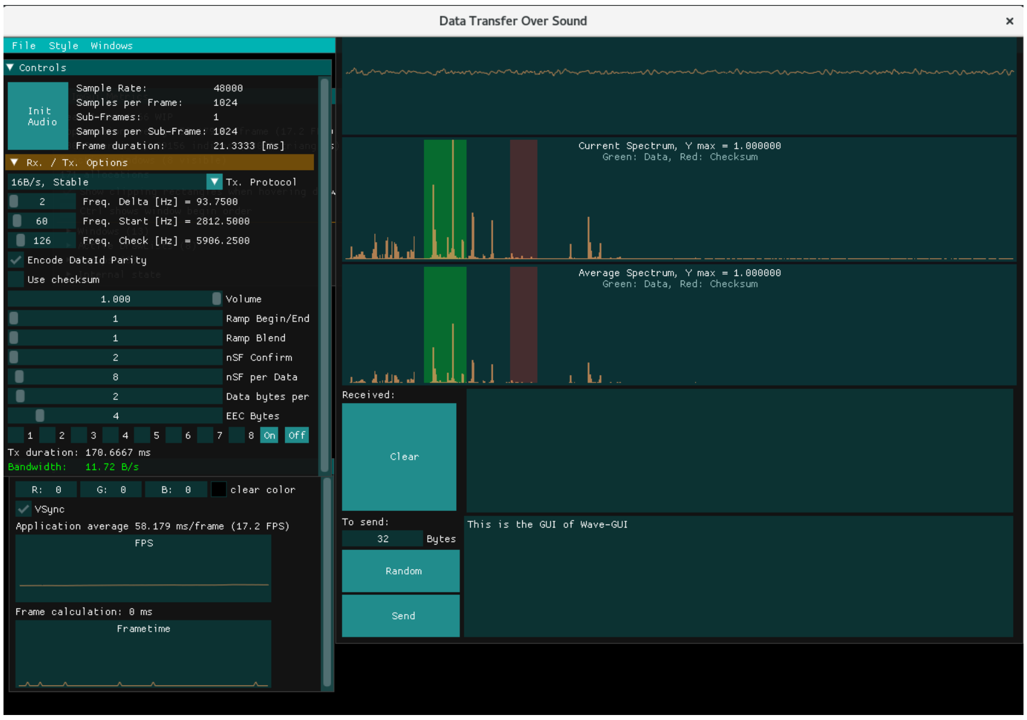
This paper presents a step-by-step procedure for implementing and evaluating an ADT application with only two computers. Section 2 provides background information on ADT and presents the Wave-GUI application, and its encoding protocols. Section 3 presents the experiment we conducted, its setup, its experimental design, and the procedure for the installation and launching of Wave-GUI. Section 4 defines several terms necessary for discussing the results of the experiment, and Section 5 presents the results of the experiment. Finally, Section 6 is the conclusion.

2 ACOUSTIC DATA TRANSMISSION

ADT transmits and receives data over the air in the form of soundwaves. In ADT communications, information is encoded into audio signals to be transmitted between speakers and microphones [4].

Since most of today's devices have a built-in speaker and microphone, we can make use of existing hardware in new ways [2]. ADT technology can support one-to-many transactions, unlike many wireless mechanisms. It has further advantages in that it is visible as an interaction media, which has been shown to increase perceived transmission rates [1].

Sonarax is a company that is working on commercially viable ADT solutions. They have unveiled a new ADT protocol which functions without the need for an Internet connection and is able to function on any device with a built-in speaker and microphone [5]. However, Sonarax is not the only company to take advantage of ADT technology. Stimshop, a company based in Paris, has introduced its own ADT protocol that can turn any speaker or sound system into a tool for wireless communication. The company claims that it can be used in virtually any circumstance [2].

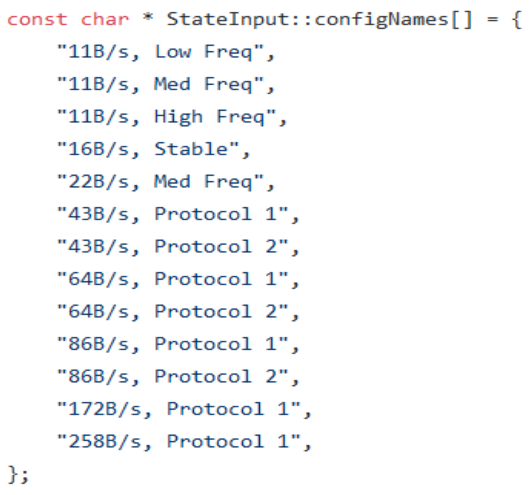

Data Transfer Over Sound
File Style Windows
Controls
Init Audio
Rx. / Tx. Options
Current Spectrum, Y max = 1.000000
Green: Data, Red: Checksum
Average Spectrum, Y max = 1.000000
Green: Data, Red: Checksum
Received:
Clear
Random
Send
FPS
Frametime

```
Sample Rate:              48000
Samples per Frame:        1024
Sub-Frames:               1
Samples per Sub-Frame:    1024
Frame duration:           21.3333 [ms]

16B/s, Stable            Tx. Protocol
2     Freq. Delta [Hz] = 93.7500
60    Freq. Start [Hz] = 2812.5000
126   Freq. Check [Hz] = 5906.2500
✓ Encode DataId Parity
  Use checksum
1.000   Volume
1       Ramp Begin/End
1       Ramp Blend
2       nSF Confirm
8       nSF per Data
2       Data bytes per
4       EEC Bytes
1 2 3 4 5 6 7 8 On Off
Tx duration: 170.6667 ms
Bandwidth:   11.72 B/s

R: 0   G: 0   B: 0   clear color
✓ VSync
Application average 58.179 ms/frame (17.2 FPS)
Frame calculation: 0 ms

To send:
32 Bytes
This is the GUI of Wave-GUI
```

Figure 1. The user interface of Wave-GUI.

Wave-GUI, shown in Figure 1, is an ADT application created by developer Georgi Gerganov. Gerganov began developing Wave-GUI in August of 2017 and finished around August of 2019. Wave-GUI is coded primarily in C++ and is dependent on several libraries. Wave-GUI uses the Graphics Library Framework (GLFW), Simple DirectMedia Layer (SDL), Fastest Fourier Transform in the West (FFTW), and ImGUI libraries [3, 6-9].

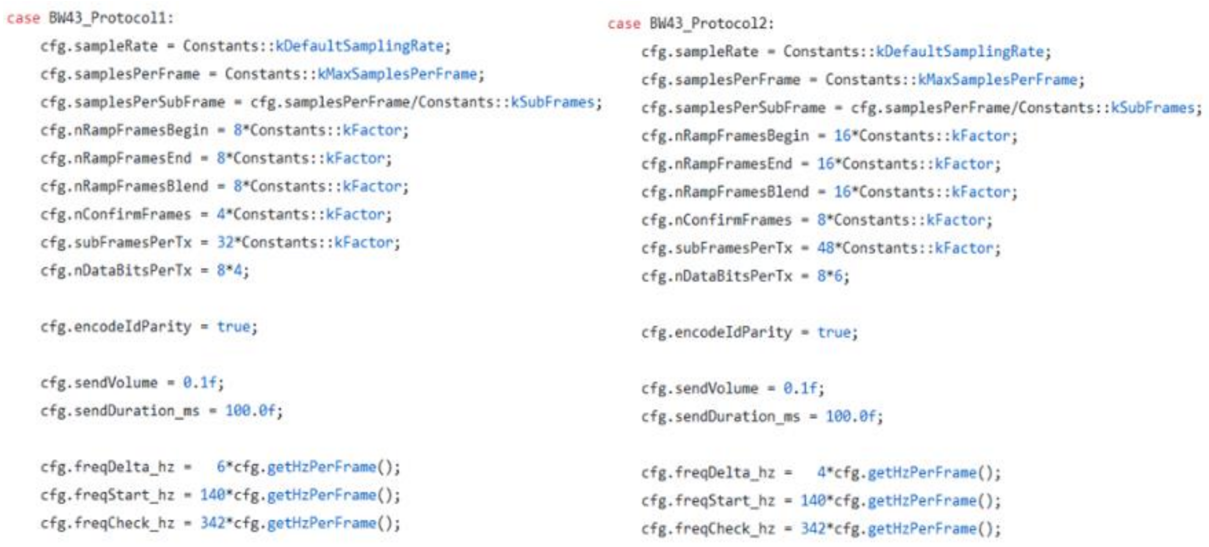
2.1 Encoding Protocols



```
const char * StateInput::configNames[] = {
    "11B/s, Low Freq",
    "11B/s, Med Freq",
    "11B/s, High Freq",
    "16B/s, Stable",
    "22B/s, Med Freq",
    "43B/s, Protocol 1",
    "43B/s, Protocol 2",
    "64B/s, Protocol 1",
    "64B/s, Protocol 2",
    "86B/s, Protocol 1",
    "86B/s, Protocol 2",
    "172B/s, Protocol 1",
    "258B/s, Protocol 1",
};
```

Figure 2. List of Wave-GUI protocols taken from the source code.

Wave-GUI allows the user to transmit data via 13 encoding protocols with varying bitrates. These include 11, 16, 22, 43, 64, 86, 172 and 258 bytes per second, or B/s. Figure 2 shows a full list of Wave-GUI's protocols as seen in the source code of the program. These protocols are distinct from one another, and they utilize a different set of Tx/Rx parameters to achieve the stated bitrate. For example, Figure 3 shows the source code for the "43B/s, Protocol 1", or "BW43_Protocol1", and "43B/s, Protocol 2", or "BW43_Protocol2", protocols [3].



```
case BW43_Protocol1:
    cfg.sampleRate = Constants::kDefaultSamplingRate;
    cfg.samplesPerFrame = Constants::kMaxSamplesPerFrame;
    cfg.samplesPerSubFrame = cfg.samplesPerFrame/Constants::kSubFrames;
    cfg.nRampFramesBegin = 8*Constants::kFactor;
    cfg.nRampFramesEnd = 8*Constants::kFactor;
    cfg.nRampFramesBlend = 8*Constants::kFactor;
    cfg.nConfirmFrames = 4*Constants::kFactor;
    cfg.subFramesPerTx = 32*Constants::kFactor;
    cfg.nDataBitsPerTx = 8*4;

    cfg.encodeIdParity = true;

    cfg.sendVolume = 0.1f;
    cfg.sendDuration_ms = 100.0f;

    cfg.freqDelta_hz =   6*cfg.getHzPerFrame();
    cfg.freqStart_hz = 140*cfg.getHzPerFrame();
    cfg.freqCheck_hz = 342*cfg.getHzPerFrame();
```

```
case BW43_Protocol2:
    cfg.sampleRate = Constants::kDefaultSamplingRate;
    cfg.samplesPerFrame = Constants::kMaxSamplesPerFrame;
    cfg.samplesPerSubFrame = cfg.samplesPerFrame/Constants::kSubFrames;
    cfg.nRampFramesBegin = 16*Constants::kFactor;
    cfg.nRampFramesEnd = 16*Constants::kFactor;
    cfg.nRampFramesBlend = 16*Constants::kFactor;
    cfg.nConfirmFrames = 8*Constants::kFactor;
    cfg.subFramesPerTx = 48*Constants::kFactor;
    cfg.nDataBitsPerTx = 8*6;

    cfg.encodeIdParity = true;

    cfg.sendVolume = 0.1f;
    cfg.sendDuration_ms = 100.0f;

    cfg.freqDelta_hz =   4*cfg.getHzPerFrame();
    cfg.freqStart_hz = 140*cfg.getHzPerFrame();
    cfg.freqCheck_hz = 342*cfg.getHzPerFrame();
```

Figure 3. The source code for the 43B/s Protocol 1 and 43B/s Protocol 2.

3 EXPERIMENT

3.1 Experiment Setup

The experiment was conducted using 2 computers: a 2015 MacBook Pro and a Lenovo P51 ThinkPad. These will be referred to as "MacBook" and "ThinkPad" for the remainder of the paper. The MacBook has a 2.5GHz quad-core Intel Core i7 CPU (Apple does not provide the specific model of the chip), 16GB of RAM, and an AMD Radeon R9 M370X GPU. The ThinkPad has a 2.9GHz Intel Core i7-7820Q CPU, 16GB of RAM, and an Nvidia Quadro M2200 GPU. Both computers are running a CentOS 8 virtual machine through the VirtualBox application. Both computers, and the instances of Wave-GUI running on them, will be set to full volume.

3.2 Experimental Design

The experiment consists of five trial sets: one control and four experimental sets. Each trial set consists of three sets of 5 transmissions per protocol (195 total transmissions per trial set). Each transmission consists of the input “Sample Data”. In total, the experiment consists of 975 discrete data transmissions.

The control trial set will be conducted with the sending computer’s speaker placed three feet away from the receiving computer’s microphone in a silent room. The MacBook will be the sending computer, and the ThinkPad will be the receiving computer.

The first and second experimental trial sets consist of one of the computers acting as both sending and receiving computer. Experimental trial set one uses the MacBook, and number two uses the ThinkPad.

The third experimental trial set is the same setup as the control trial set, but with a white noise machine playing at full volume placed in between the sending and receiving computers. The fourth experimental trial set replaces the white noise with thunderstorm noises coming from the same machine also set to full volume.

3.3 Installation

The software was installed on a CentOS 8 virtual machine running on VirtualBox. The first step in this process is installing “CMake GUI”. This is done via the CentOS software application. The next steps are to install “Development Tools”, ensure that PowerTools is enabled, and install epel. Run the following commands to do this:

```
sudo yum install “Development Tools”

sudo yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-
8.noarch.rpm

sudo dnf config-manager –set-enabled PowerTools
```

Next, the dependencies must be installed. Run the following command:

```
sudo yum install SDL2 SDL2-devel glfw glfw-devel fftw fftw-devel libXi-devel
libXinerama-devel libXrandr-devel libXxf86vm-devel libXcursor-devel
```

The Guest Additions VirtualBox software must be installed to properly run Wave-GUI. Run the following commands:

```
sudo dnf install epel-release

sudo dnf install gcc make perl kernel-devel kernel-headers bzip2 dkms
```

The virtual machine must then be rebooted, and the Guest Additions CD Image must be input into the virtual cd drive. It is found in the VirtualBox dropdown menu at the top of the virtual machine window. Once installed, the virtual machine must be rebooted once again. Finally, Wave-GUI is installed via the following commands:

```
git clone https://github.com/ggerganov/wave-gui

cd wave-gui

git submodule update –init

make
```

If everything was installed correctly the Wave-GUI software should now be ready to launch.

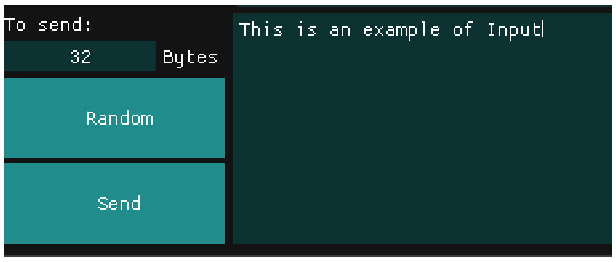
3.4 Launching Wave-GUI

To launch Wave-GUI, open a terminal window and input the following commands:

```
cd wave-gui

./build/main/wave-gui
```

4 DEFINITIONS

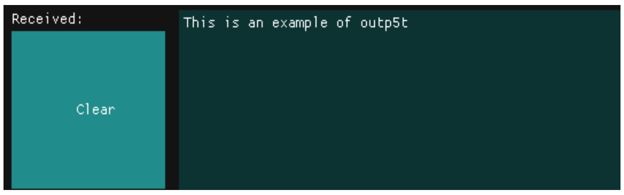
Several important terms must be defined to properly understand the results. Input is the text typed by the user into the input box to be transmitted by Wave-GUI. Figure 4 shows the input box populated with the phrase "This is an example of Input".



This is an example of Input

Figure 4. Input Displayed in the input box.

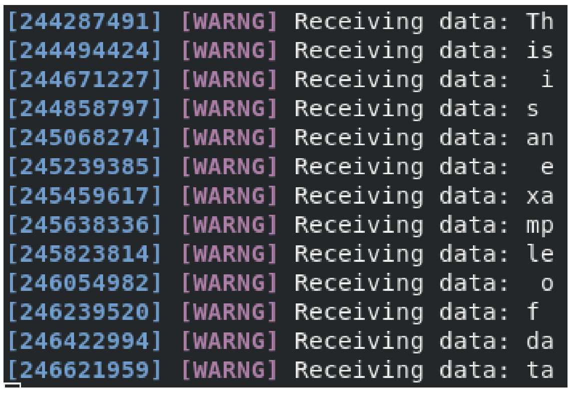
Output is the text displayed in the output box. Any transmission that Wave-GUI picks up is displayed as output. Figure 5 shows an example of output.



This is an example of outp5t

Figure 5. Output shown in the output box.

Data is defined as the raw information that Wave-GUI interprets from the sounds that the computer's microphone picks up. Frames are the lines of data that make up what was input. The transmission shown in Figure 6 is made up of 13 frames of data that make up the phrase "This is an example of data". Both data and output can be split into frames.



```
[244287491] [WARNG] Receiving data: Th
[244494424] [WARNG] Receiving data: is
[244671227] [WARNG] Receiving data:  i
[244858797] [WARNG] Receiving data: s
[245068274] [WARNG] Receiving data: an
[245239385] [WARNG] Receiving data:  e
[245459617] [WARNG] Receiving data: xa
[245638336] [WARNG] Receiving data: mp
[245823814] [WARNG] Receiving data: le
[246054982] [WARNG] Receiving data:  o
[246239520] [WARNG] Receiving data: f
[246422994] [WARNG] Receiving data: da
[246621959] [WARNG] Receiving data: ta
```

Figure 6. Wave-GUI data shown in terminal log.

5 RESULTS & FINDINGS

The data gathered from the experiment is summarized in tables 1, 2, and 3. The criteria columns show various metrics by which the data has been measured. The "% Some Output" and "% No Output" columns denote the percent of trials that did or did not have any output. A high percentage is better for the "% Some Output" criterion, and a low percentage is better for the "% No Output" criterion. The "% Perfect Output" column denotes the percent of trials where the output was the exact same as the input. A high percentage is better for this criterion. The "% Imperfect output" column denotes the percent of trials where the output was not the exact same as the input. A low percentage is better for this criterion The "% Single Frame" column denotes the percent of trials where the output consisted of a single frame, and the "% Multiple Frames" column denotes trials where the output consisted of multiple frames. A high percent is better for the "% Single Frame" criterion, and a lower percent is better for the "% Multiple Frames" criterion.

Table 1 shows the percent of trials that match each of the criteria for protocols and for trial sets. The averages for the protocols and the trial sets were the same. Table 2 shows the percent of trials that match each of the criteria for the trial sets. Table 3 shows the percent of trials that match each of the criteria for the protocols.

Table 1. Data Averages.

Criteria / Data Set	% Some Output	% No Output	% Perfect Output	% Imperfect Output	%Single Frame	%Multiple Frames
Protocols	76	24	38.05	61.95	60	16
Trial sets	76	24	38.05	61.95	60	16

Table 2. Data Averages by Trial Set.

Criteria / Data Set	% Some Output	% No Output	% Perfect Output	% Imperfect Output	%Single Frame	%Multiple Frames
Control	68.72	31.28	14.87	85.13	34.36	34.36
MacBook	91.79	8.21	60	40	89.23	2.56
ThinkPad	100	0	86.15	13.85	100	0
White Noise	54.36	45.64	12.82	87.18	35.38	18.97
Storm	65.13	34.87	16.41	83.59	41.03	24.1

Table 3. Data Averages by Protocol.

Criteria / Data Set	% Some Output	% No Output	% Perfect Output	% Imperfect Output	%Single Frame	%Multiple Frames
11B/s, Low Freq	53.33	46.67	17.33	82.67	49.33	4
11B/s, Med Freq	56	44	28	72	53.33	2.67
11B/s, High Freq	61.33	38.67	18.67	81.33	58.67	2.67
16 B/s, Stable	62.67	37.33	26.27	73.33	50.67	12
22 B/s, Med Freq	66.67	33.33	30.67	69.33	56	10.67
43 B/s, Protocol 1	64	36	42.67	57.33	64	0
43 B/s, Protocol 2	100	0	36	64	60	40
64 B/s, Protocol 1	86.67	13.33	46.67	53.33	65.33	21.33

64 B/s, Protocol 2	100	0	34.67	65.33	52	48
86 B/s, Protocol 1	100	0	41.33	58.67	65.33	34.67
86 B/s, Protocol 2	100	0	42.67	57.33	68	32
172 B/s, Protocol 1	92	8	89.33	10.67	92	0
258B/s, Protocol 1	45.33	45.33	40	60	45.33	0

5.1 Trial sets

The ThinkPad and MacBook trial sets consistently had the best rankings for each of the criteria. As shown in Table 4, the ThinkPad trial set always had the highest numbers for the positive criteria ("% Some Output", "% Perfect Output", and "% Single Frame") and the lowest numbers for the negative criteria ("% No Output", "% Imperfect Output", and "% Multiple Frames"). The MacBook trial set was right behind the ThinkPad trial set for each of these criteria. They were close, but how close depended on the criteria. The MacBook trial set trailed the ThinkPad trial set by only 2.56% for the "% Single Frame" criteria but lagged behind by 26.15% for the "% Perfect/Imperfect Output" criteria.

Table 4. Rankings per Criteria per Trial Set.

Criteria Data Set	% Some Output	% No Output	% Perfect Output	% Imperfect Output	%Single Frame	%Multiple Frames
Control	3	3	4	4	5	5
MacBook	2	2	2	2	2	2
ThinkPad	1	1	1	1	1	1
White Noise	5	5	5	5	4	3
Storm	4	4	3	3	3	4

The control, white noise, and storm trial sets consistently trailed the ThinkPad and MacBook trial sets. The introduction of exterior noise for the white noise and storm trial sets was expected to affect the data transmission much more than it had.

The white noise, storm, and control trial sets jumped around in position depending on the criteria. The white noise trial set was in last place for two thirds of the criteria, while the storm trial set was the highest of the three for two thirds of the criteria.

The results of the trial sets suggest that Wave-GUI's performance depends on two factors: the hardware and microphone-speaker proximity. The MacBook and ThinkPad trial sets show that even in identical situations the microphone and speaker hardware being different can have a big impact on the reliability of the transmission.

The control, white noise, and storm trial sets show that microphone-speaker proximity is also an important factor. The introduction of extra environmental noise did have an impact on the transmissions, but accurately predicting how much of an impact this noise might have would prove difficult.

5.2 Protocols

The initial testing of Wave-GUI using the online demo tool suggested that the lower bitrate protocols would be more robust and produce more accurate outputs than the higher bitrate protocols. This was not the case. As shown in Table 5, the protocol that consistently ranked the highest was "172B/s Protocol

1.” It had the best ranking for 2/3 of the criteria. We believe this was because the higher bitrate encoded more data into fewer frames, and thus took less time. This made the transmission less susceptible to environmental noise due to the lower exposure time.

Table 5. Rankings per Criteria per Protocol.

Criteria / Data Set	% Some Output	% No Output	% Perfect Output	% Imperfect Output	% Single Frame	% Multiple Frames
11B/s, Low Freq	12	12	13	12	12	6
11B/s, Med Freq	11	11	10	9	9	4
11B/s, High Freq	10	10	12	7	7	4
16 B/s, Stable	9	9	11	11	11	8
22 B/s, Med Freq	7	7	9	9	8	7
43 B/s, Protocol 1	8	8	3	3	5	1
43 B/s, Protocol 2	1	1	7	7	6	12
64 B/s, Protocol 1	6	6	2	2	3	9
64 B/s, Protocol 2	1	1	8	8	10	13
86 B/s, Protocol 1	1	1	5	5	3	11
86 B/s, Protocol 2	1	1	3	3	2	10
172 B/s, Protocol 1	5	5	1	1	1	1
258B/s, Protocol 1	13	13	6	6	13	1

The ratio of “% Perfect Output” trials to “% Some Output” trials is another good metric for the performance of each protocol. The ratio shows the performance of each protocol in an ideal situation in which every transmission is detected and interpreted in some way. The ratios are shown in Table 6. This ratio shows that “172B/s, Protocol 1” is the best performing with a ratio of .971, followed by”258B/s, Protocol 1” at .882. The third-place protocol, “43B/s, Protocol 1”, trails by .215 points, with a ratio of .667. The worst performing protocols were the high and low 11B/s protocols. The high and low frequency protocols had ratios of .304 and .325, respectively. These were the worst and second worst ratios. Interestingly, the medium frequency 11B/s protocol had a ratio that was close to the median with a ratio of .5.

Table 6. Ranking per Protocol of the Ratio of “% Perfect Output” to “% Some Output”.

Criteria / Data Set	Ratio of “% Perfect Output” to “% Some Output”	Rank
11B/s, Low Freq	.325	12
11B/s, Med Freq	.5	5
11B/s, High Freq	.304	13
16 B/s, Stable	.426	8
22 B/s, Med Freq	.46	6
43 B/s, Protocol 1	.667	3
43 B/s, Protocol 2	.36	10

64 B/s, Protocol 1	.539	4
64 B/s, Protocol 2	.347	11
86 B/s, Protocol 1	.413	9
86 B/s, Protocol 2	.427	7
172 B/s, Protocol 1	.971	1
258B/s, Protocol 1	.882	2

6 CONCLUSIONS

The results of the experiment show that higher bitrate protocols tend to perform better than lower bitrate protocols. It also showed that the performance of acoustic data transmission is highly dependent on the hardware used to send and receive the data. Although some of the protocols performed poorly, they all produced output more than half the time. Despite this, Wave-GUI is not ready to be used in a real-world network environment as the best performing protocol was only able to produce output 89% of the time. For reference, a packet loss rate between five and ten percent would significantly affect a VoIP call [10].

We must consider, however, that this paper's experiment took place entirely within the range of human hearing, and that many of the ADT applications in use utilize ultrasound for data transmission [2]. If other use cases are considered, Wave-GUI may be a suitable competitor to Bluetooth. Chirp, a similar application to Wave-GUI, has been compared to Bluetooth in other papers, and users generally preferred the ADT application partially due to the aesthetic of the sound [1]. Though some parts of its software can be a bit rough around the edges (the installation process comes to mind), Wave-GUI is a well-rounded piece of software that showcases various methods of acoustic data transmission..

REFERENCES

[1] Mehrabi, A., Mazzoni, A., Jones, D. et al., "Evaluating the user experience of acoustic data transmission.", Pers Ubiquit Comput 24, 655–668 (2020). https://doi.org/10.1007/s00779-019-01345-7

[2] Chandler, Simon. "How Data-Over-Sound Will Ensure A Permanently Connected IoT World." Forbes, Forbes Magazine, 21 Oct. 2019, www.forbes.com/sites/simonchandler/2019/10/18/how-data-over-sound-will-ensure-a-permanently-connected-iot-world/?sh=2141af9c6343.

[3] Gerganov, Georgi. "Ggerganov/Wave-Gui." GitHub, github.com/ggerganov/wave-gui, Retrieved Aug 10, 2019

[4] Nesfield, James. "Sending Data Over Sound: How and Why?" StackPath, 24 June 2019, www.electronicdesign.com/industrial-automation/article/21808186/sending-data-over-sound-how-and-why.

[5] Joseph-Raz, Limor. The Challenges of Developing Data Over Sound Technology, 2 Jan. 2020, www.sonarax.com/post/why-its-so-challenging-to-develop-data-over-sound-technology-why-the-other-protocols-failed.

[6] "GLFW - An OpenGL Library." GLFW, www.glfw.org/, Retrieved April 8, 2021.

[7] "About SDL." Simple DirectMedia Layer, www.libsdl.org/.

[8] M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3," in Proceedings of the IEEE, vol. 93, no. 2, pp. 216-231, Feb. 2005.

[9] Ocornut. "Ocornut/Imgui." GitHub, github.com/ocornut/imgui, Retrieved July 7, 2021.

[10] Mansfield, K.C. & Antonakos, J.L. (2010). Computer Networking from LANs to WANs: Hardware, Software, and Security. Boston: Course Technology, Cengage Learning. p. 501.