CZESTOCHOWA UNIVERSITY OF TECHNOLOGY

FACULTY OF MECHANICAL ENGINEERING AND INFORMATICS

ENGINEER DIPLOMA WORK

APPLICATION FOR DATA VIA SOUND TRANSPORTATION AND METHODS OF HOW TO IMPROVE TRANSMISSION QUALITY

Aplikacja służąca do przekazywania danych za pomocą dźwięku i metody poprawy jakości transmisji

Moiseienko Mykyta

Album number: 125459

Direction: Computer science

Specialization: Software engineering

Studies mode: stationary

Studies level: I

Promotor: Ph.D. Krystian Lapa

Date of work acceptation:

Promotor's signature:

Częstochowa, 2019

To Erasmus of Rotterdam, an inspiration for the lifetime experience

List of contents

Introduction

This chapter contains general information about the topic related to this work, including desired achievements, history of the project and it's practical use.

Project background

DataViaSound, DataViaSound project – reference to this diploma project.

Initially, the name of this project was used by the author when he started implementation. History of DataViaSound starts in October 2016. On course 'Introduction to telecommunication applications and services' author received a task to create arbitrary application which would perform data transfers between two devices. Author participated in this course during a semester in University of Maribor (Slovenia). He studied in UM via Erasmus+ program (it was author's 3rd semester).

Idea of the application came into author's mind when he'd tried the system of student coupons in Slovenia. Every student of Slovenian university or school could benefit from a daily discount (2.6 euros) in slovenian restaurants. In order to use a coupon, student had to call the special phone number. Then (still while call is in process) a mobile phone had to be put close to dedicated Margento terminal (which looks like the terminal for bank cards payment; see Img. 1 Use of Margento terminal). Then the phone made a unique sound (noise) which came from the phone call. This noise contained encoded data which was needed to identify a user of a coupon. In next step, Margento terminal sent request to server for verification of a user. If server returned positive response, the terminal would show that use of a coupon was allowed. Later, restaurant received money compensation from Slovenian state for each used coupon.

Img. 1 Use of Margento terminal

Impressive aspect of the system was the precision of sound-to-data decoding. Even in crowded and noisy room Margento terminal managed to identify a user of a coupon in 1-2 seconds of listening to message which was encoded in sound. Normal distance between a terminal and phone was around 5 cm.

While details of implementation of Margento system were not well known, there were some obvious points:

- Sound recorders built into Margento terminals should be of good quality;
- Ways of communication:
  - *phone – mobile operator – server – Margento terminal;*
  - sound producer (phone) – sound recorder (terminal);
- Phone sends repeated sound messages to Margento terminal.

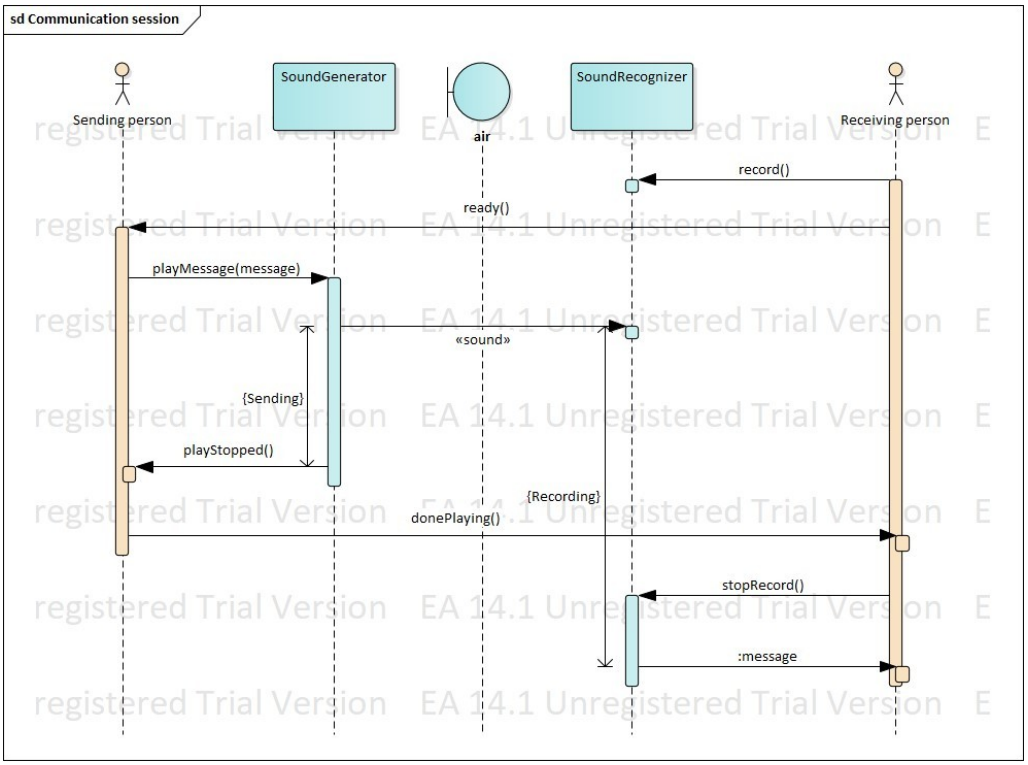
Commercial potential of the solution

Author designed the application expecting that the applied way of communication could be used in future as a hardware independent analog of NFC. NFC was a data transfer technology, which has been introduced for first time in Samsung Nexus S in 2010 [3], and was mainly used in smartphones since then. Problems of NFC were that it only works:

- With special chip embedded inside of the smartphone. DataViaSound can work on any devices that have a microphone and loudspeaker (WiFi, Bluetooth, etc. also aren't required). So in fact it was a completely hardware independent solution, since every mobile device has loudspeakers and microphones.

- When devices physically contact with each other. DataViaSound, potentially, can work if devices are located on the opposite sides of a room (or any other finite space). On the other hand, close physical contact ensures that communication session happens exactly between chosen devices. More advanced solution of DataViaSound would require familiar exclusiveness of data exchange.

Basic use case of DataViaSound on example of one communication session:



Img. 2 Activity diagram of communication session between the sender and receiver

Aims of the project

On the beginning of this diploma project, author had defined some target achievements, which he wanted to gain during development of the project. Some of the aims were successful, others were partially successful. There were also failed aims, because they required more work (eventually in master diploma project).

Below you can find a complete list of initial aims:

- [PRIORITY AIM] High accuracy even on microphones of low quality (popular problem of mobile devices), also in condition of noisy surrounding. Partially achieved.
- [PRIORITY AIM] Create a tool for automated testing and evaluation of different transportation algorithms (DataViaSoundTester v0.1). Client-server application. Perform continuous automated tests with use of 1 Windows PC (server) and 2 Android smartphones (clients), all connected within one Wi-Fi network. Server collects test results, stores them as statistics and guarantees synchronization of clients' actions. Achieved.
- [SECONDARY AIM] Make sound more friendly for human ears. Not achieved.
- [SECONDARY AIM] Find a way to put more data in smaller frames. Not achieved.
- [SECONDARY AIM] Fix some bugs of DataViaSound v0.1 and improve stability. Partially achieved.
- [RESTRICTION] No second way of communication should used, only *sound sender → sound receiver relation.* Feedback from the receiver would be also acceptable, but only using the same communication channel – sound (useful for implementation with repeated message redundancy – receiving device would need to signalize when message no longer needs to be repeated).

1 Theoretical concepts

Chapter Theoretical concepts contains an overview on general ideas, tools and technologies related to the project. Also, it contains definitions related to DataViaSound project generally. Subsection 2.2.1 Terminology related to testing environment contains specific definitions related to DataViaSoundTester project.

1.1 Definitions (terminology)

Below you can find definitions of concepts which occur throughout this work.

Sources - source code, resources and configuration files; programmable part of the project

Android Studio – developer tool for PC (by Google) which allows to create end Android applications.

Android project view - mode of project explorer in Android Studio version 3.0 (Project → Android) which allows to see project's folders and files in structured way.

Activities, activity - a Java code that supports actions which take place on screen views or UI. In other words, a building block of the user interface. [1]

UI - user interface.

Layout description - file in .xml format which describes structure of view displayed on screen during work of application.

Transportation session – process of data sending which involves next steps:

- production of beeps by sender;
- recording of beeps by receiver;
- further decoding of beeps by receiver.

START_TAG, END_TAG – sequence of beeps which are played on start and end of each message; used by receiver algorithm for localization of time when useful message begins and ends in audio record.

Enclosing tags – START_TAG and END_TAG.

Screen view – result of navigation in Android application. What user sees on display of the smartphone.

Sound producing unit – loud speakers or headphones, part of Android device.

Android device – mobile phone with Android OS installed.

Android OS – operating system which was developed by Google and installed on wide range of vendors of mobile phones. DataViaSound application was developed only for this OS.

Java class, class – structural unit of Java project, which encapsulates it's state (as fields) and provides it's behavior (as methods).

IDE – integrated development environment.

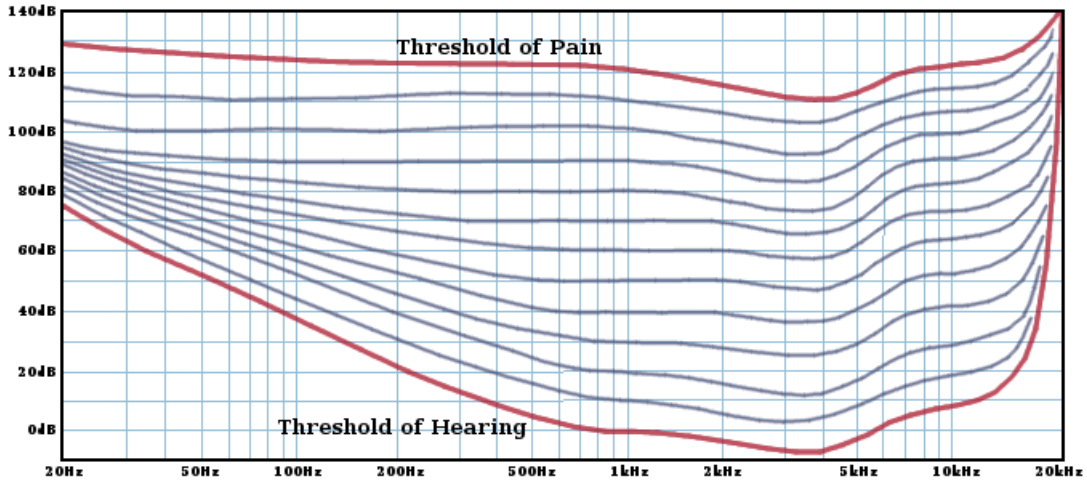
1.2 Introduction to sound

Sound – physical phenomenon, which is created by vibrations in surrounding. The surrounding has to be of density higher than zero. In everyday life, people usually perceive the sound from air which surrounds them. The main characteristics of the sound:

- Frequency (in Hz) – a period of harmonic sine wave which denotes the vibration. Multiple harmonic sine waves can cumulate at the same time. People can perceive sound in 20 – 20000 Hz spectrum.
- Loudness (in dB) – amplitude of cumulated sine wave which denotes the vibration.

In common models of smartphones, microphones and loudspeakers are designed in a way that the ranges of operable frequencies don't exceed the spectrum of frequencies accepted by human ear. Wider range of frequencies is redundant and can potentially generate higher cost of hardware production.

Loudness perception by human is not linear regarding frequency. Meaning, the threshold of hearing (loudness level when sound can be distinguished) and the threshold of pain (loudness level when sound becomes uncomfortable for human) wary depending on the frequency: an average person can distinguish 1 kHz harmonic sound starting from 0 dB, the sound becomes uncomfortable starting from 120 dB; meanwhile, the same person can distinguish 200 Hz sound starting on 30 dB, the sound becomes uncomfortable starting from 123 dB (see Img. 3 Fletcher–Munson curves: how frequency-pitch relation changes perception of the loudness by human [2]).



Img. 3 Fletcher–Munson curves: how frequency-pitch relation changes perception of the loudness by human [2]

1.3 Description of used tools and technologies

1.3.1 Java

Java is an object-oriented language which is based on classes, currently developed and supported by Oracle Corporation. Initially, it was developed by James Gosling, Mike Sheridan and Patrick Naughton. Java 1.0 was published by Sun Microsystems in 1996. Back then, developers expected it to be used mainly for television. A couple of years later they have seen a good potential of this language in Web environment. [4]

Nowadays, this language is deeply integrated in our everyday life – wide range of smartphones, Web servers, desktop applications utilize Java. It was designed with cross-platform support in mind, meaning that it should run on most of the popular operating systems. This effect is achieved by compiling a source code first into interim byte code instead of straight into machine code. This byte code is then translated into machine code by platform-specific Java Runtime Environment.

Oracle doesn't have monopoly on decisions about future of Java specification, instead Java Comunity Process organisation is responsible for it. Multiple technical companies and independent programmers from all over the world are working on new versions Java specification. [5]

1.3.2 Android

Android is an operation system based on Linux kernel and dedicated mainly for mobile devices. It is developed and maintained by Google corporation, SDK v1.0 was introduced

in 2008 [6]. Since author was familiar with Java on the beginning of his work on this project, Android was a good choice as primary platform for DataViaSound application. Android is mainly based on Java, with Android API as extension. Java SE API is still available for developers on Android, also Java syntax is preserved.

1.3.3 Sony Xperia Z5

Sony Xperia Z5 was author's smartphone which he used in everyday life and for automated tests during development of this work. It was approx. 1 year old since the date of purchase, had two loudspeakers and two microphones located on front top and front bottom sides of a body [7], 2900 mAh battery, CPU Snapdragon 810 [8], Android version 7.1.1. Geekbench multicore performance result: 3500 – 4000 [9].

1.3.4 Lenovo A1000

Lenovo A1000 was author's smartphone which he used only for automated tests during development of this work. It was approx. 2 years old since the date of purchase, had one loudspeaker on top front and one microphone on bottom front sides of a body, 2000 mAh battery, CPU Quad-core 1.3 GHz Cortex-A7 [10], Android version 5.0. Geekbench multicore performance result: 900 – 1000 [11].

1.3.5 Libraries applied for automated testing

- GSON v2.8.5 – library for easier work with json file format. Enables serialization and deserialization of objects of complex Java classes. Developed by Google. Used in DataViaSound too.
- PvdBerg1998/PNet v1.5.10 – wrapper library for TCP protocol. Author's motivation of using this library: no intention to waste time for implementation of TCP wrapper. Implementation of TCP protocol (Socket class) in JDK 1.8 didn't provide a reliable way to check if connection is alive. PNet library seemed to support this check. All-in-all, connection-alive check was implemented in completely different manner in DataViaSoundTester, so PNet was not needed anymore. But removal of this library would require significant amount of time for rewriting client-server code. Used in testing module of DataViaSound too.
- JFreeChart v1.0.13 – library for creating visual charts programmatically. In DataViaSoundTester it was used for creating line charts for test results.

2 DataViaSound communication

DataViaSound application, Application - Android application dedicated to this diploma project. Mentions with a version (eg. “v0.1”) are also considered as references to this definition.

Chapter DataViaSound communication is divided into three sections. Section 2.1 describes early version of the application, it’s algorithms, problems and ideas of improvement. Section 2.2 depicts development of the tool for automated testing and analysis of results of performed tests. Section 2.3 contains a detailed list of changes introduced to the application during this diploma project.

Target operating system of the project – Android, programming language – Java.

2.1 DataViaSound v0.1

DataViaSound v0.1 – initial version of the application. Work on version 0.1 within course at University of Maribor was started in November and finished in December. Further improvements of the project were implemented in DataViaSound v1.0 and described in subsection 2.3 DataViaSound v1.0 – improvements and bugfixes.

2.1.1 Structure of project files (in term of Android project view)

Location of application sources in folder structure of the project:

DataViaSound → app → src → main

File tree:

- **app**
  - **manifests**
    - AndroidManifest.xml
  - **java**
    - **com.luckynick.android.test**
      - GetFrequencyHandler
      - IterateForFrequenciesHandler
      - MainActivity
      - ProjectTools
      - SoundGenerator
      - SoundRecognizer
  - **res**
    - **layout**
      - activity_main.xml
      - detect_text_layout.xml

- one_frequency_layout.xml
- play_message_layout.xml
- record_layout.xml

■ **menu**
- basic_menu.xml

■ **mipmap**
- ic_launcher.png

■ **values**
- 600-2000_hex.xml
- colors.xml
- strings.xml
- styles.xml

App – root folder for sources.

File **AndroidManifest.xml** from **manifests** folder configures Android application: declares needed permissions for access to system features, sets name of application, makes references to style of UI, declares activities which exist within this application.

Java source code of application is stored in package **com.luckynick.android.test** from **java** folder. Description of classes comes further in this document.

Folder **res** (resources) contains layout descriptions (res → **layout**, res → **menu**), graphical image files (res → **mipmap**) and constant values (res → **values**). Constant values can be accessed from Java code using object R (e.g. `R.array.`***`frequencies`***).

2.1.2 Frequencies binding

Frequencies binding – array where index represents encoded character and value represents frequency in Hz which corresponds to this encoded character. Each symbol which can be transferred using this application, has one assigned frequency to it. Size of array equals to number of characters in chosen character encoding scheme (example: ASCII contains 128 characters). Value *-1* means that symbol isn't used in data-to-sound encoding and decoding, no frequency is assigned to it. Thus, attempt to send this character will result in silence during corresponding frame.

File \res\values**600-2000_hex.xml** was used for mapping *ASCII* symbols to sine wave frequencies (frequencies binding) from *600* to *2100* Hz. Value *-1* means that symbol isn't used in data-to-sound encoding and decoding. Symbols 0-9 and A-F are used in this frequency binding file, so it was suitable for using hexadecimal numbers in data encoding-decoding.

Example of .xml file with frequencies binding (exerpt from 600-2000_hex.xml):

```
<integer-array name="frequencies">
    <item count="0">-1</item>
    <item count="1">-1</item>
    <item count="2">-1</item>
        …
    <item count="48">600</item>
    <item count="49">700</item>
    <item count="50">800</item>
    <item count="51">900</item>
        …
    <item count="126">-1</item>
    <item count="127">-1</item>
</integer-array>
```

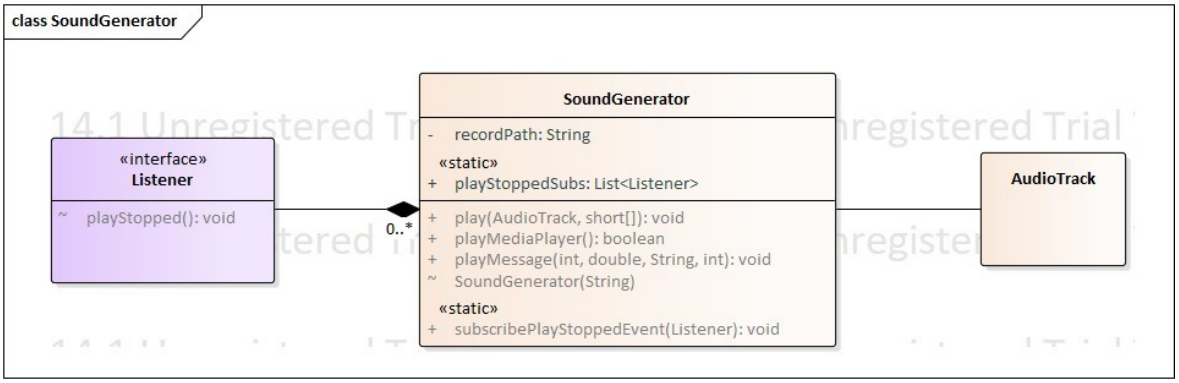
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Img. 4 Table of ASCII symbols

2.1.3 Algorithms of sender

Class **SoundGenerator** (SoundGenerator.java)

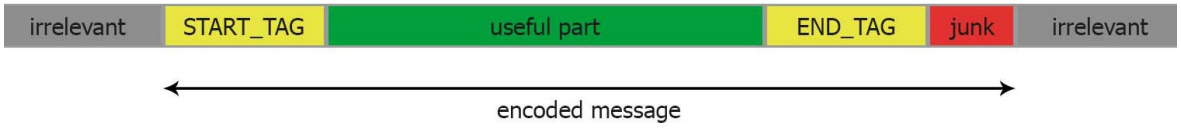


Img. 5 Diagram of SoundGenerator class

Sender – first participant of transportation session, device which generates a message encoded into beeps. Sound is played through the sound producing unit.

Data message – entire data which is sent during transportation session.

Encoded message – sequence of frames, result of data-to-frequencies translation. On low level it is raw audio data. Includes redundancies: START_TAG, END_TAG, junk part.



Img. 6 Structure of an encoded message

Description of the above illustration: An encoded message consists of:

- Enclosing start and end tags – sections of audio record, which consist of several beeps. They signalize a start or ending of the section which contains useful data;
- Useful part (section) – section of record, which contains data in frequency encoded state. Only this part was used for actual sound-to-data decoding.
- Junk section – small (a couple of beeps) part of audio record, always located in the ending of whole encoded message. It was used because of (eventually) a bug in some speakers of Android devices, which leads to distortion in last beep during production of beeps by sender. This causes failure during localizing of end tag.

Audio data – array of numeric values which represent original analog signal (samples); used by sound producing unit.

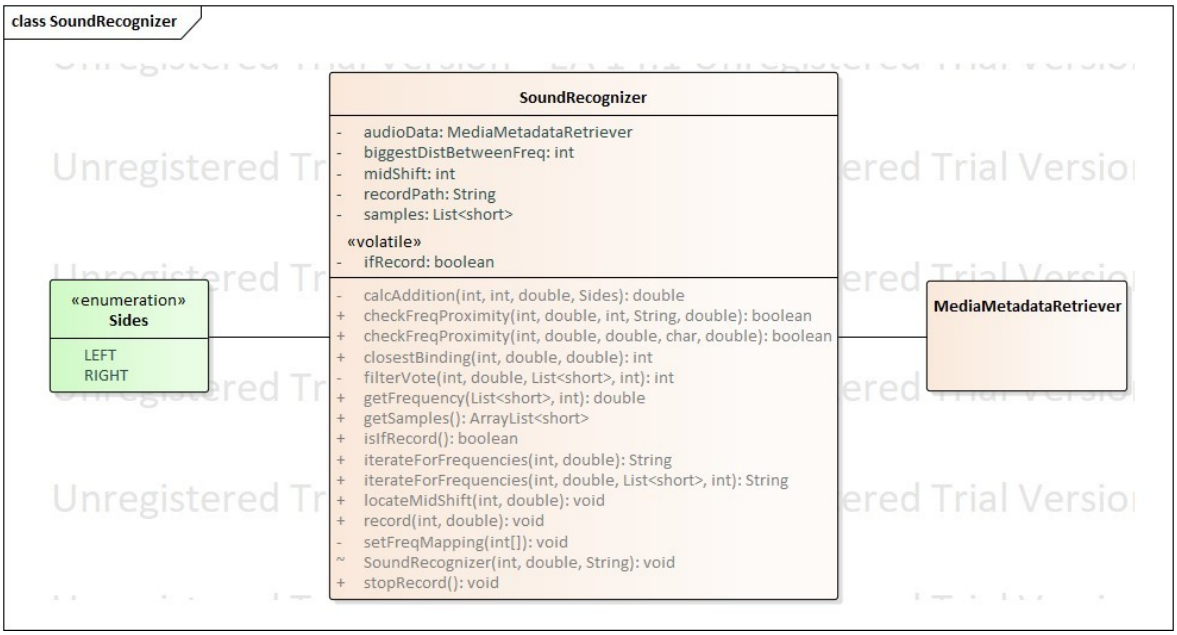
Steps for data sending:

- Transform data message into HEX numbers.

- Translate symbols (0-9 and A-F) which represent HEX numbers into corresponding frequencies.
- Generate raw audio data which contains harmonic sine waves. Frequencies of those sines correspond to positions of frames in which they are contained.
- Pass audio data to the sound producing unit. It results in production of beeps. Perception of continuous similar sound, which characteristics are denoted by predefined time length and same frequency is called beep. A beep is a direct match to concept of frame with only difference: a beep is a result of perception of sound by human, but frame is a part of raw audio data which is processed by electronic device. All characteristics of a beep are inherited from the frame. Frequency of beep directly corresponds to the location of its frame in an encoded message. Duration is constant and predefined. Beeps altogether represent sequential pieces of data in a transmission-ready message. Result of passing of an encoded message to sound producing device.

2.1.4 Algorithms of receiver

Class **SoundRecognizer** (SoundRecognizer.java)

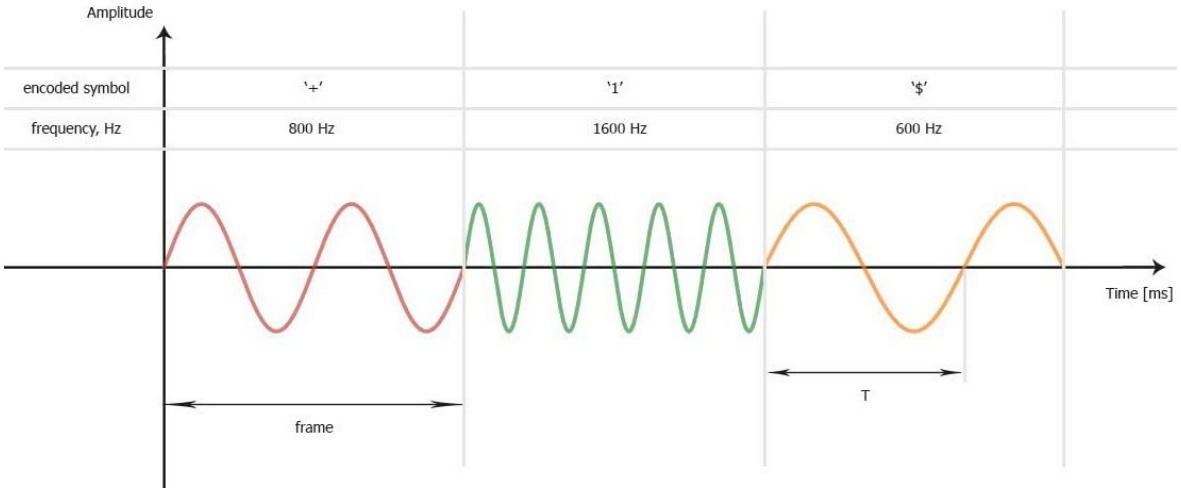


Img. 7 Diagram of SoundRecognizer class

Receiver – second participant of transportation session, device which catches beeps created by sender and decodes data message which is stored in those beeps.

Frequency modulation (FM) – approach which is used in systems which involve signal processing. Involves use of varying wave frequencies for transportation of data. The first version of the application was based on the ideas of FM.

Device records sound which contains sines of different frequencies. Example input is illustrated on Image 8.



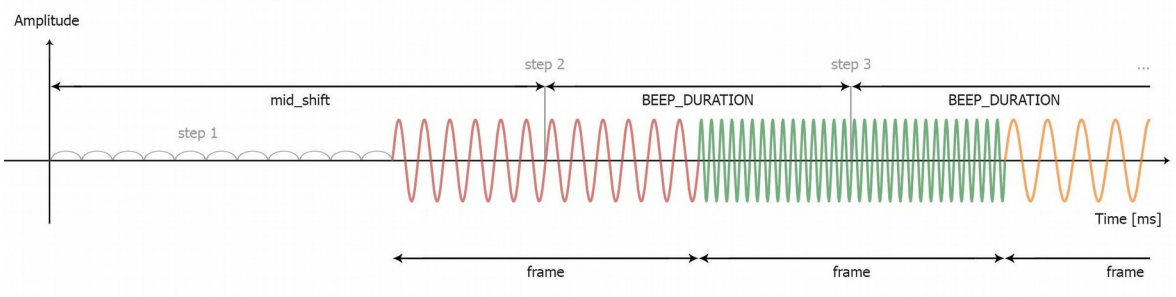
Img. 8 Example input

Description of the above illustration: input is a continuous sound composed of sound frames. Every **frame** contains simple sine wave, denoted by predefined **constant time length** and **frequency** which is specific to the encoded symbol. Each symbol which can be transferred using this Application, has one assigned frequency to it. For example, a symbol '+' is assigned to frequency 800 Hz. This means that during transmission of this symbol, sine wave with frequency 800 Hz and time length of a frame will be generated by a sending device; receiving device will analyze frame and detect that it contains sine wave of frequency 800 Hz, thus receiver knows that this frame represents symbol '+'.

2.1.4.1 Walking algorithm

This is an algorithm which regulates steps of how SoundRecognizer class decodes audio record and how corresponding decoding pointer moves through this record. It results in frequency values for each beep frame contained in audio record. Further, those frequencies are translated into data using frequencies binding. This algorithm was created by the author.

Decoding pointer – numeric value which is used during process of getting frequencies from an audio record in the sequential way. This value is incremented. Frequency is counted on time position denoted by this pointer.



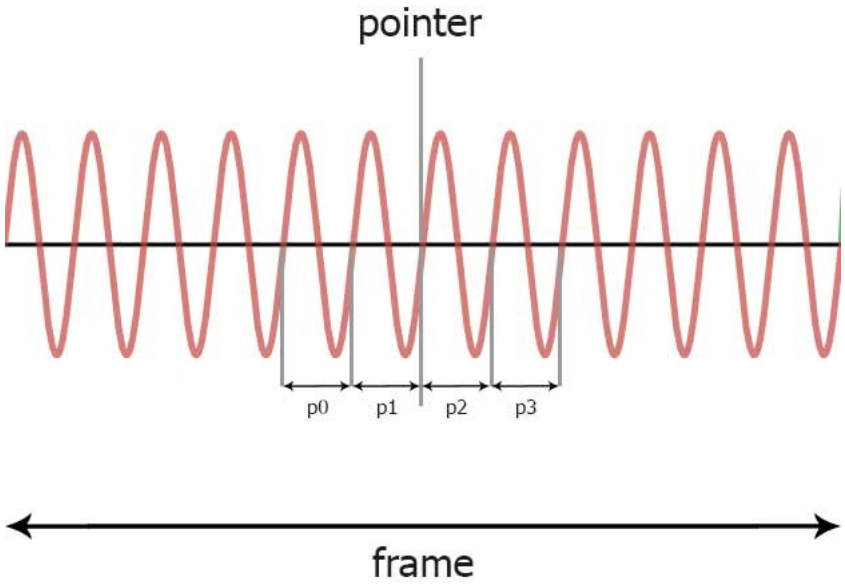
Img. 9 High level illustration of walking algorithm

Given the audio record, which contains continuous sequence of harmonic sine sounds (frames) of constant time duration (BEEP_DURATION), it is possible to count frequencies on arbitrary time point of audio record. Correct value will be returned only if beep is caught by a receiver in audio record. It means that receiver, by applying own algorithms of frequency counting, can associate frequency in a given frame with any frequency in frequencies binding. -1 will be returned otherwise. We need to be able to count frequencies on different time points in order to decode source message from received audio record. This is achieved by acquiring sequence of frequencies. Further, frequencies are translated into characters (HEX numbers in v0.1) using frequencies binding. After that, 2 HEX symbols are used for getting one ASCII character. A complete message is ready when all characters are transformed from HEX to ASCII.

Steps of walking algorithm on high level:

- Record audio from a microphone;
- Set decoding pointer on beginning of audio record;
- Make small iterations (10 ms) of the pointer (see Step 1, Img. 9) and count frequency value (see Img. 10):
    - If frequency value doesn't equal to frequency of the first symbol in START_TAG, then continue iterations
    - Otherwise, check if next frames contain rest of symbols in START_TAG. If true, set mid_shift = pointer + BEEP_DURATION * 0.5. Now mid_shift points to center of first frame (see Step 2, Img. 9);

- Make normal iterations (BEEP_DURATION ms) of the pointer and count frequency value (see Step 3, Img. 9 for pointer incrementation and Img. 10 for frequency counting);
- Stop when the pointer reaches to the end of an audio record. It means that END_TAG is not taken into account for now. Instead, it will be used for localization of useful part after frequency-to-data translation.



Img. 10 Illustration of surrounding principle in frequency counting

Let's review function for counting frequency on one specific point of time (low level walking algorithm). Next steps are performed during counting of frequency:

- Set pointer which counts frequency on given time
- Count **n** lengths of sines which surround the pointer, divide each of them by bitrate → we get frequencies in surrounding of a given pointer. In conditions when there is no noise around, frequency values are equal. But this situation is impossible in reality.
- Having an array of frequencies for surrounding of the pointer, algorithm should decide what is the dominant frequency in this surrounding. There are two decision making approaches:
  - most frequent occurrence among frequencies (used in stable v0.1)
  - average of frequencies (author's subjective opinion: this approach works worse)

- Resulting dominant frequency is considered as actual frequency on pointer position

Surrounding of pointer position is taken into account because we can't rely on only one frequency value during count of frequency. Noises can distort frequency on that one point of time, so frequency value can be inaccurate.

2.1.4.2 Translation between data message and transmission-ready message

Transmission-ready message – result of a transformation from data message by applying a transformation algorithm. Use of a transformation is not compulsory, depends on implementation. If no transformation is applied to data message, then it is considered as transmission-ready message. Transformation algorithm is used in v0.1 for reducing number of frequencies' assignment, which results in bigger frequency range available for each symbol in frequencies binding. Manual tests showed that bigger frequency range reduces error rate during transportation sessions (subjective opinion, which is not supported by statistics; during development of v0.1 there was no tool for creating statistics).

Transformation algorithm in v0.1 translates message data into HEX numbers which come from ASCII table. So in fact beeps contain only symbols which are used for representation of HEX numbers (0-9 and A-F).

A receiver, after getting all frequencies from audio record, performs reversed transformation algorithm: translates symbols which represent HEX numbers back to data message.

2.1.5 Rest of the code

ProjectTools (ProjectTools.java) class contains constants and utility methods which are used globally in the application.

MainActivity (MainActivity.java) class is an activity which controls behaviour of all views and is responsible for initialization after a launch of the application.

- Asynchronous tasks (eg. Play audio record) are launched from here
- Clicks on items of screen menu are handled here
- Clicks on buttons of screen views are handled here
- Constructors of SoundGenerator and SoundRecognizer classes are called from here

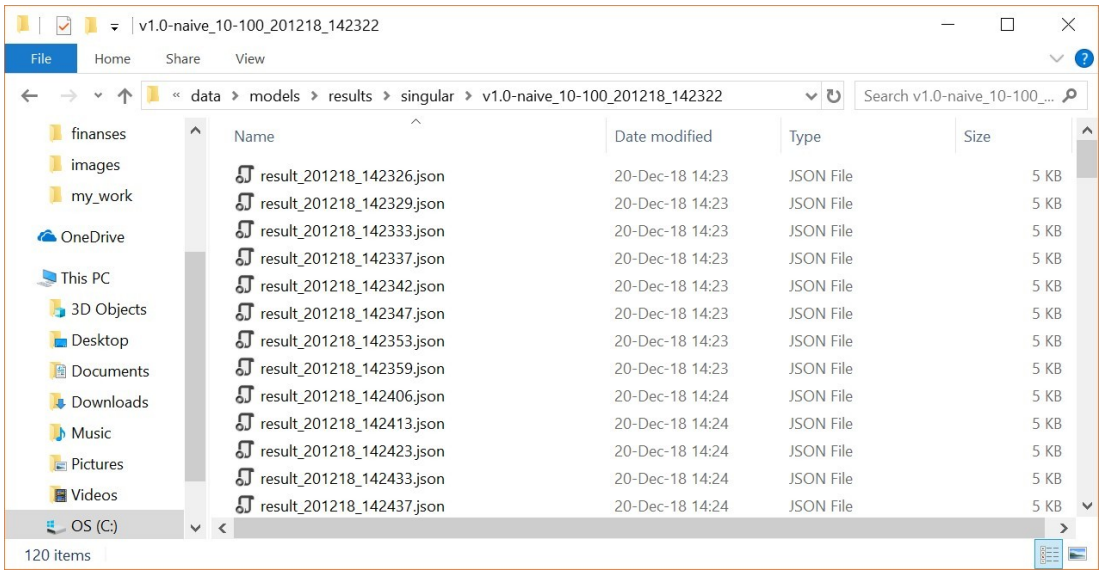
2.1.6 Bugs and improvements of version 0.1

1. [BUG FIXED] 'F' symbol has no assigned frequency in frequencies binding, thus it is not played. It caused 8 of 128 ASCII symbols to be processed with error by application.
2. [BUG FIXED] Algorithm of finding midShift doesn't work properly (doesn't find a middle of first beep correctly). It happened because the algorithm was finding earliest countable frequency which value is lower than frequency of 'E' symbol, which was wrong approach. The new algorithm finds START_TAG, this approach results in correct localizing of midShift. This bug usually didn’t affect work of a program, but in rare situations receiver demonstrated bad results, since pointer was set on a border of two frames.

2.2 DataViaSoundTester

DataViaSoundTester – tool, which was created specially for performing automated tests on DataViaSound Android application.

The main reason of why author decided to create this tool: there was no way of how to determine if some changes bring positive or negative impact on work of application. Also, there was a need to study work of application under different conditions (noise in surrounding, loudness level of produced beeps, etc.). Such testing tool would measure the success level of communication sessions between devices. Result of performed tests sequence would be a line chart (plot). Form of chart is convenient for analysis of large data sets.



Img. 11 Results (logs) of the tests are stored in folder data → models → results


```
{
  "senderSessionSummary": {
    "summarySource": {
      "macAddress": "48:88:CA:F2:03:5E",
      "localIP": "192.168.43.1",
      "vendor": "LENOVO",
      "model": "Lenovo A1000",
      "os": "Android",
      "osVersion": "5.0",
      "roleOfParticipant": "PEER",
      "isHotspot": true,
      "nameOfModelClass": "Device",
      "filename": "device_null_null_201218_142324.json",
      "wholePath": "data/models/devices/device_LENOVO_Lenovo A1000_201218_151649.json",
      "customPrefix": ""
    },
    "sendParameters": {
      "soundProductionUnit": "LOUD_SPEAKERS",
      "loudnessLevel": 10,
      "message": "az",
      "frequenciesBindingShift": 900,
      "frequenciesBindingScale": 1.0
    },
    "sessionStartDate": 1545315408897
  },
  "receiverSessionSummary": {
    "summarySource": {
```

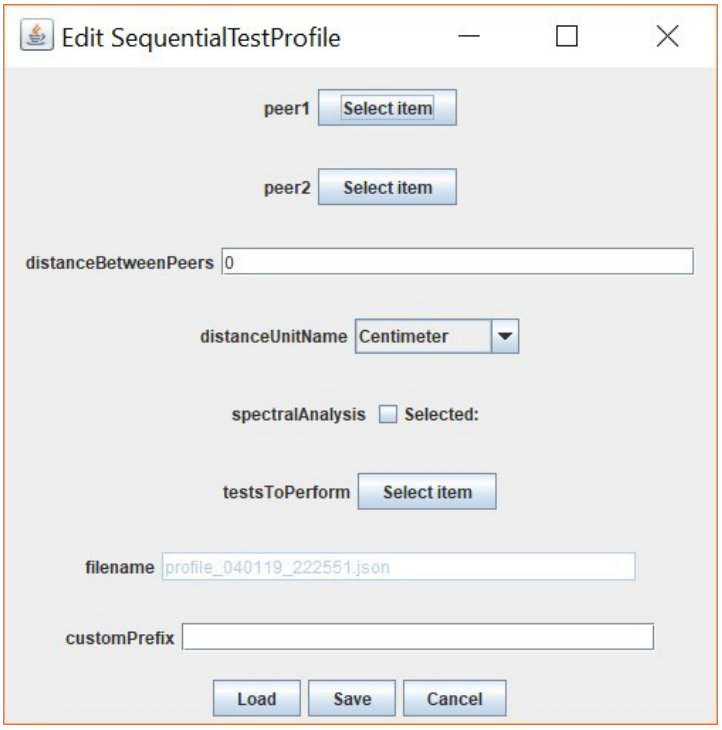

Img. 12 Test results are stored in JSON format

DataViaSoundTester program enables a user to create test profiles, run multiple tests which don't require human participation (according to test profiles created earlier), show results of previously performed tests.

For the sake of simplicity, menu navigation is done in console interface.

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
Arguments list: --createProfile
Select kind of action to perform:
1. CreateSingularProfile
2. CreateSequentialProfile
3. CreateDictionary
4. CreateConfig
5. CreateCumulatedReport
Input int value from 1 to 5: 2
```

Img. 13 Interface of the testing tool is partially console based



Img. 14 Screenshot of the window which enables an operator to create a profile of tests sequence

Elements which define quality of data transportation via sound channel:

- Low error rate;
- High bandwidth (speed; 5 bytes/second in v0.1);
- Acceptation by human ear, sound should be less irritating.

Error rate is critical among above elements, so improvement of other elements shouldn't harm error rate in the first place. Tests analysed in subsection 2.2.5 Analysis of existing cumulated test reports were performed with error rate reduction in mind, since it was the main problem of DataViaSound v0.1.

2.2.1 Terminology related to testing environment

Success level – percent representation of how received message is different from sent message durring single test, where 100% success means sent and received messages are equal. The main metric of a test. How two strings are compared:

- Find the longest character sequence (basic match) which matches in both strings.
- Starting from position of last symbol of basic match found in previous step, compare characters in each incremented position. If characters are not equal – match percentage decreases.
- If received message is longer than sent message – match percentage is decreased proportionally: match_percentage * sent.length() / received.length()

Examples of string comparison:

- sent "ala ma kota", received "do tego ala ma kota dużego", match: 42% (received string was much longer).
- sent "ala ma kota", received "do tego bla mfgkota dużego", match: 30% (received string was much longer and distorted).

Dictionary – set of strings which are sent between devices during a single test.

Single test – sequence of communication sessions (in both directions; peer1 → peer2 and peer2 → peer1), in which data is represented by elements of the associated dictionary. Single test is configured in object of SingleTestProfile class. Result of a single test is put into object of SingleTestResult class. Attributes of a single test:

- Loudness level – integer from 0 to 100. Used by sending peer for determining the loudness of produced beeps. Same loudness level won't result in same actual loudness level produced by devices. Reason: parameters of hardware (loudspeakers) vary on different models of devices.
- Dictionary – data which is sent during a test. There is a separate communication session for each element of a dictionary. Same element of a dictionary is sent in both directions.
- Frequencies binding shifts – list of integers. Each value of this list shifts all frequencies binding values by its value. Single test is repeated for a single element

of this list. With this property set, test will give valid result only if containing sequential test has spectral analysis property set to true.

- Frequencies binding scale – this property can be set to predefined double values which correspond to specific frequency bindings on application side. With this property set, test will give valid result only if containing sequential test has spectral analysis property set to true.

Sequential test – array of single tests. Sequential test is configured in object of SequentialTestProfile class. Result of a sequential test is put into object of SequentialTestResult class. Any existing sequential test profile can be set as default in configuration.json file and can be run repeatedly specific amount of iterations. SequentialTestProfile holds data about:

- Two peer devices (Device class) which participate in this sequence of single tests.
- Distance between peers – information for manual analysis of test results.
- Spectral analysis – boolean value which replaces default way of how tests are performed. In this mode, messages are sent not only on different loudness levels, but also in different spectrums (ranges) of frequencies. This mode enables us to find the best frequencies spectrum for DataViaSound application.
- List of single tests. Each SingleTestProfile from this list is performed in a sequential way.

Device – object which holds information about a participant of a test. Fields of Device class:

- Mac address – relatively unique identifier of a device in local area network. Used as ID of a device during tests.
- Vendor, model, OS family and version – information for manual analysis of test results.

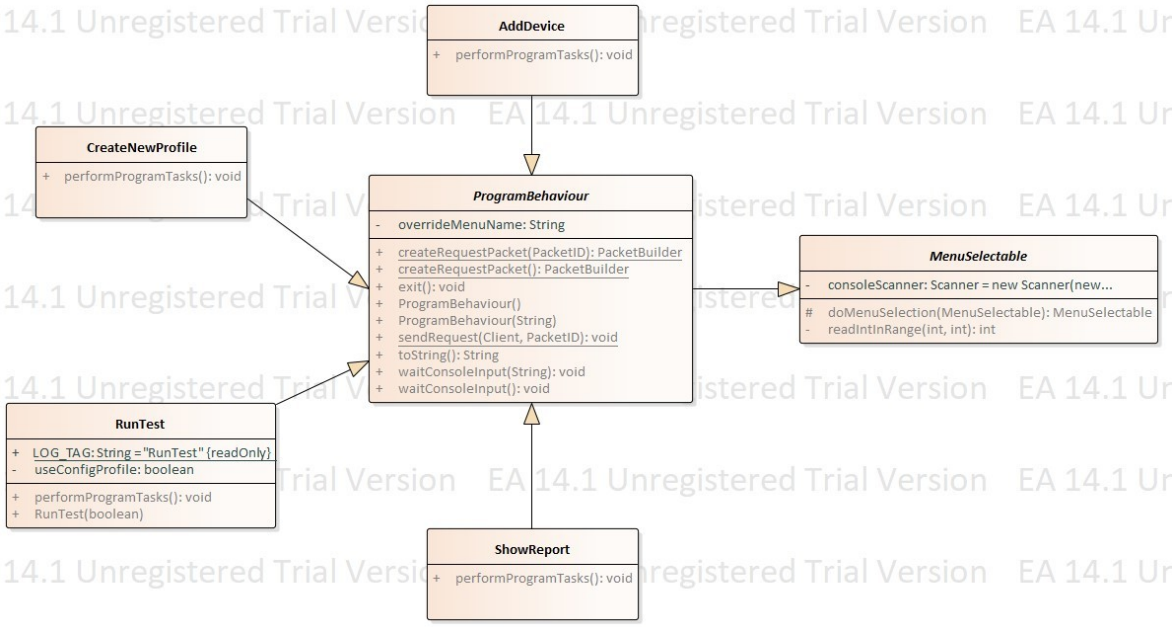
Cumulated report – aggregation of arbitrary amount of sequential test results. Results of all tests in each sequential test are summed up using the same loudness levels and messages as IDs for usual sequential tests, loudness levels and frequency binding bases as IDs for spectrum analysis tests.

Test operator – person who configures and starts a sequence of tests, probably always author of this diploma.

Control test – performed in "perfect" conditions. Its result is used for comparison with other results of tests. It has defined attributes: frequency binding, noise of surrounding, relative positions of devices, required number of sequential test reports. Currently, report on control test has to contain 8 sequential test reports. Static attributes of control tests (valid for all):

- No noise around
- 8 sequential tests are performed
- Devices stay horizontally on a platform, their displays face each other, on the distance of platform's length

2.2.2 Source code

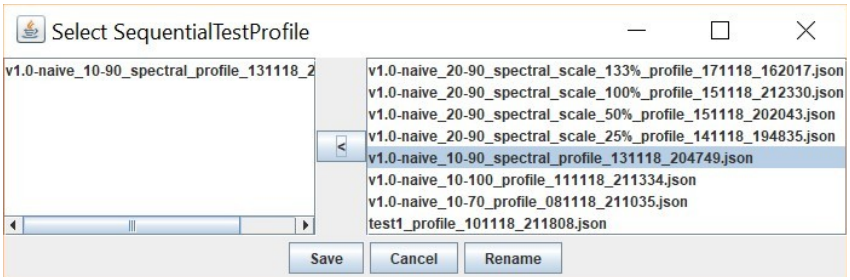


Img. 15 Diagram of classes in package com.luckynick.behaviours

Above diagram shows classes which enable certain behaviors of the testing program:

- AddDevice class – first registration of a device for future use in tests. Waits for devices to connect to the same network and traps it's data if it is ready to send it (DataViaSound application should be running and be in test mode on this device).

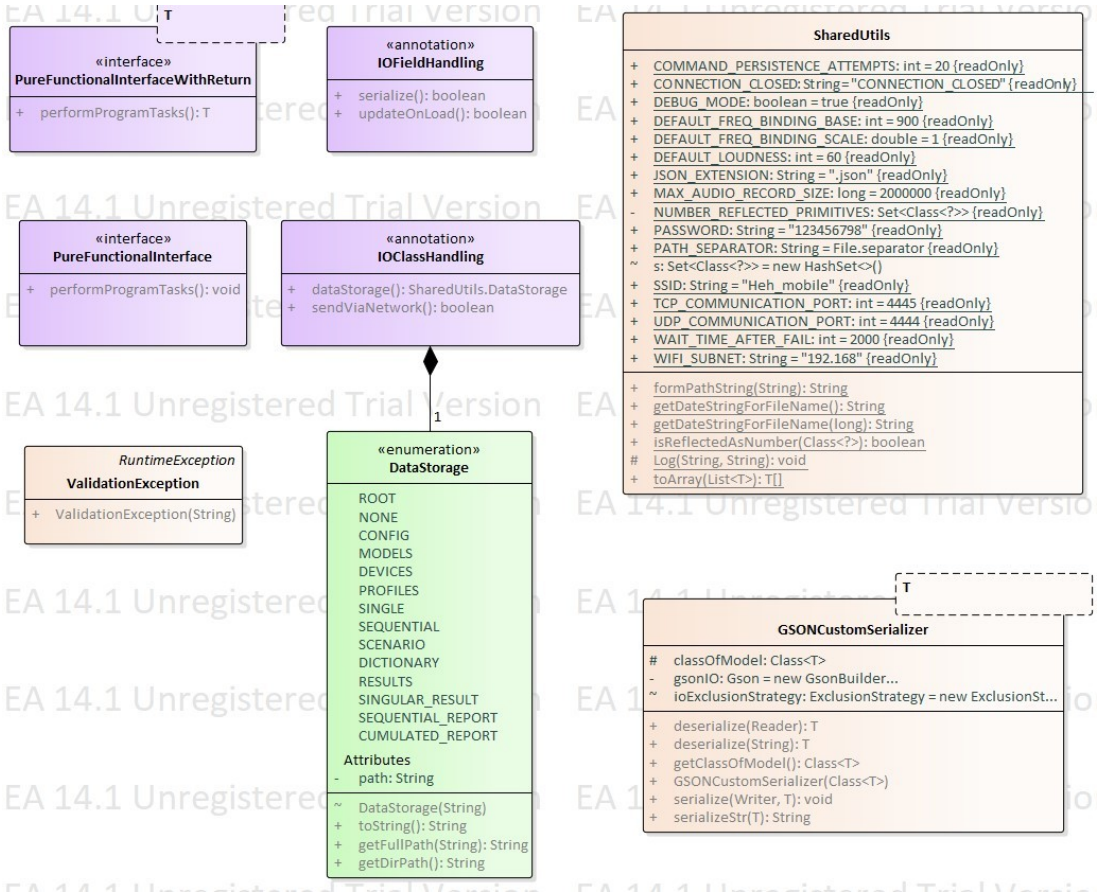
- CreateNewProfile – create a profile of specific class. Opens window with fields which operator can edit. When operator is done with filling in the fields, the profile will be saved.
- RunTest – perform sequence of tests using existing sequential test profile. Opens a window with list of existing instances of SequentialTestProfile class stored in folder data\models\profiles\sequential or runs the one from configuration.



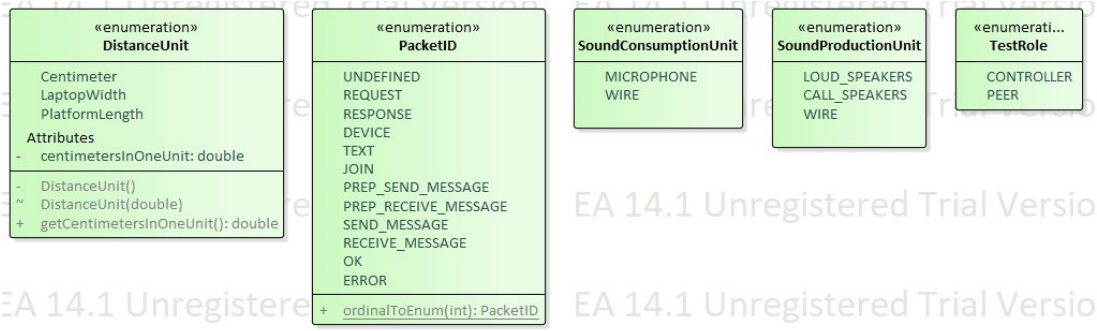
Img. 16 Selection of existing sequential test profile before running a test

- ShowReport – prompts a class of a report from an operator, opens a window with list of existing instances of selected class stored in folder data\models\results and, after selection of report instance, it is shown in a new window as charts and text description.

Any of those behaviors can be invoked by passing corresponding argument during start of the program. Some of the behaviors require further menu navigation using console interface. For example, CreateNewProfile behavior will prompt an operator to choose a class of a profile which he desires to create.

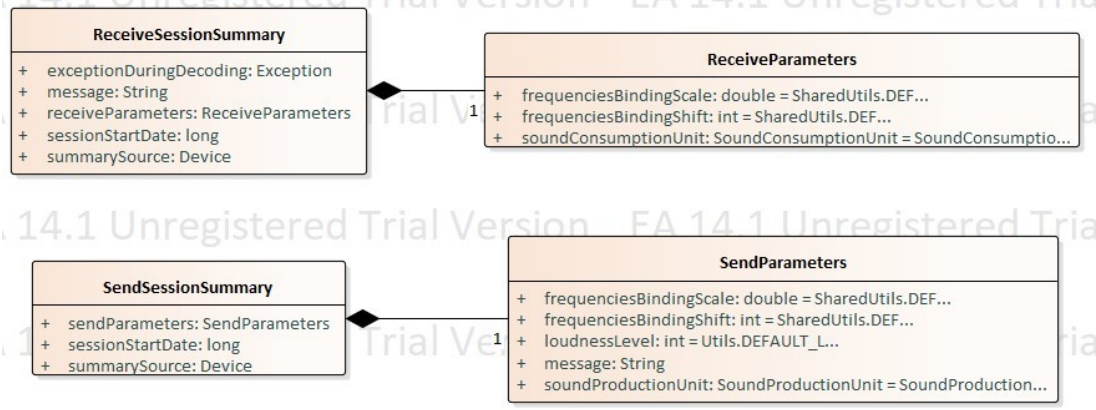


Img. 17 Diagram of classes in package com.luckynick.shared



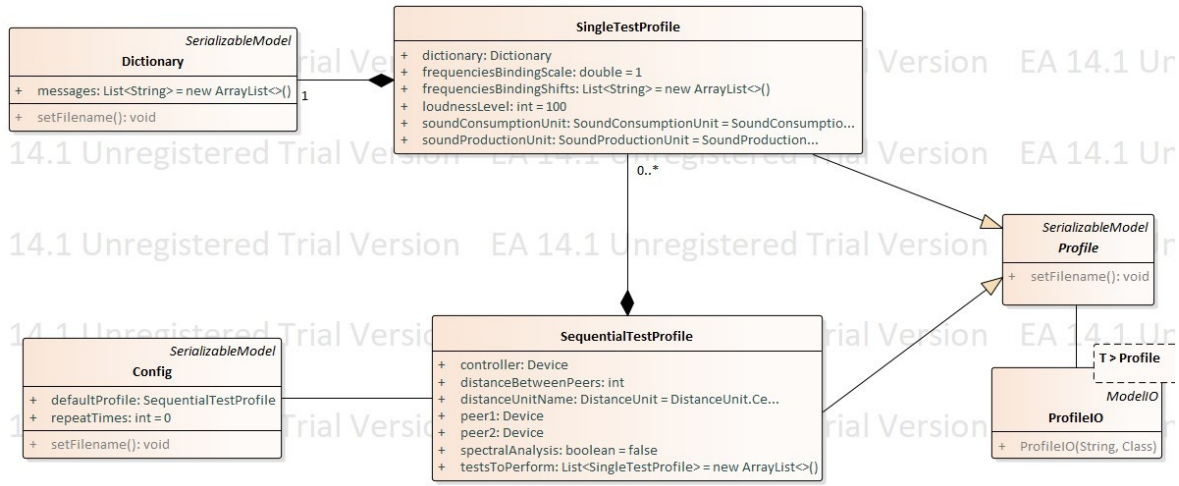
Img. 18 Diagram of classes in package com.luckynick.shared.enums

PacketID enum from the above package is used for indication of packet type when it is received by a party of communication within one testing network. Other enums are used in test profiles and reports.



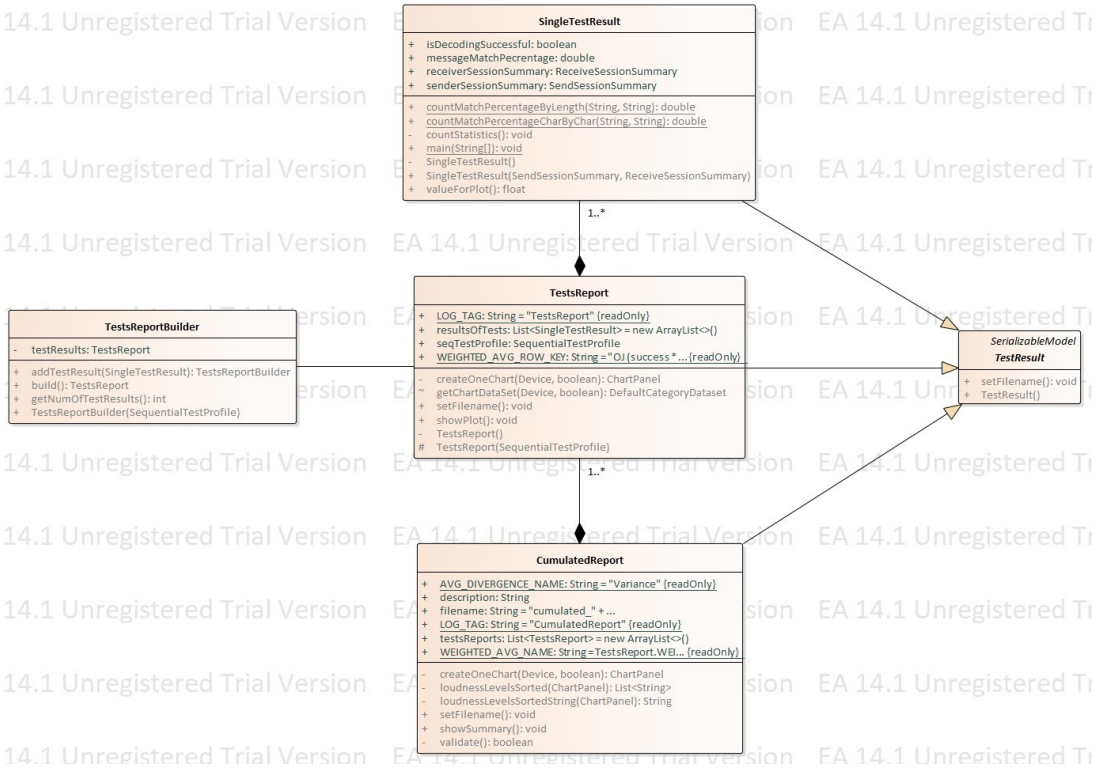
Img. 19 Diagram of classes in package com.luckynick.shared.model

Structure of the package shown above contains java classes shared with DataViaSound application for purposes of tests. It means that DataViaSound project contains identical copy of this package. Since some classes are required by both projects in the same way, it was reasonable to put them in one shared package. For example, classes which contain parameters or results of communication should be identical on both peer and controller of a test (see Img. 19). Also, some global variables should be shared (see SharedUtils class on Img. 17).



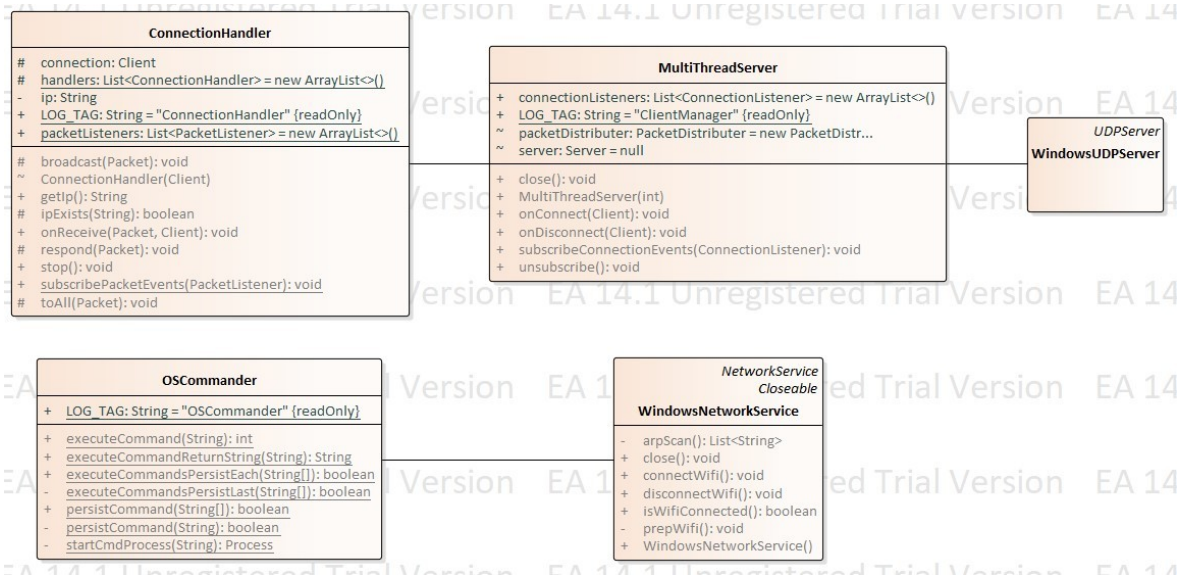
Img. 20 Diagram of classes in package com.luckynick.models.profiles

Package shown on above diagram contains profiles which are used during tests. An operator also can create a configuration which holds an instance of sequential test profile and can run it multiple times, making a process of testing fully independent from an operator. Any instance of Profile class or its subclasses can be serialized using an instance of ProfileIO class.



Img. 21 Diagram of classes in package com.luckynick.models.results

After test is executed, its results are serialized in folder data\models\results. A whole bunch of collected single test results is stored in tests report. Later multiple tests reports can be joined in one cumulated report.

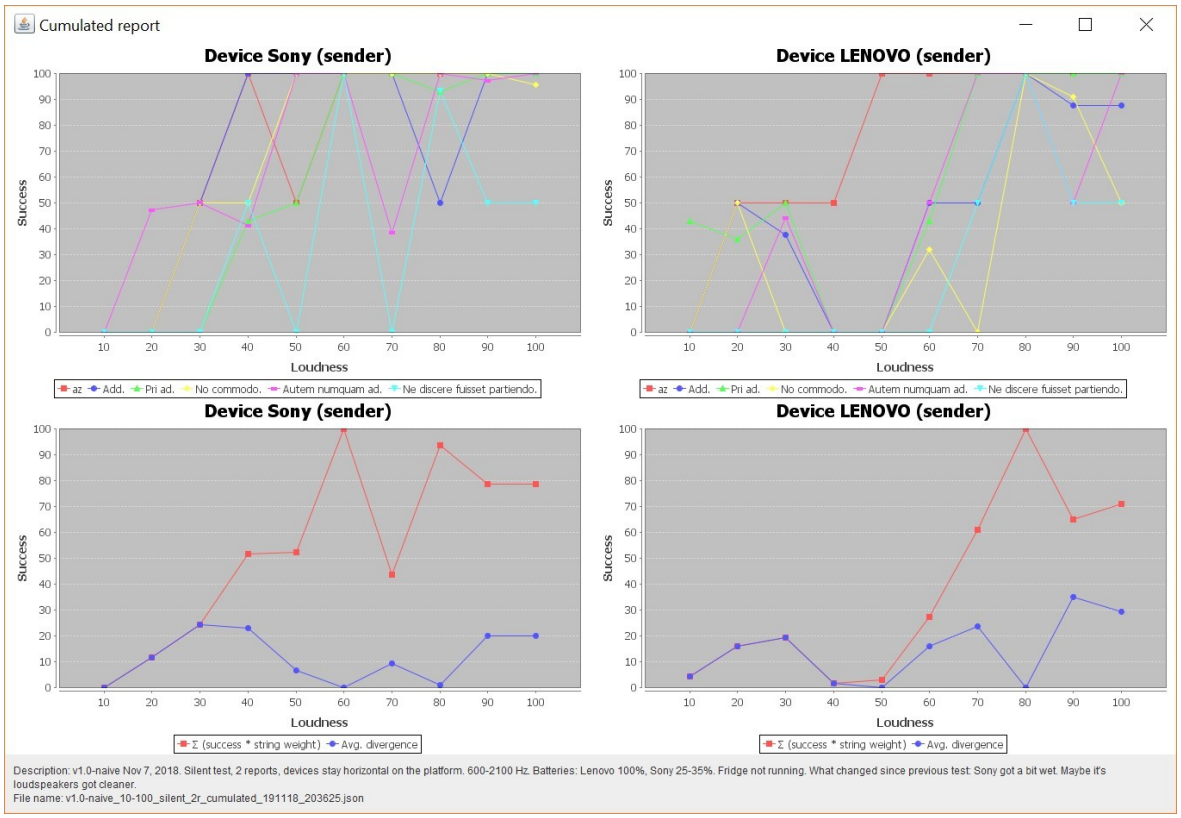


Img. 22 Diagram of classes in package com.luckynick.net

Above diagram shows classes of the package which is responsible for communication within common Wi-Fi network dedicated for testing. OSCommander is an interface to

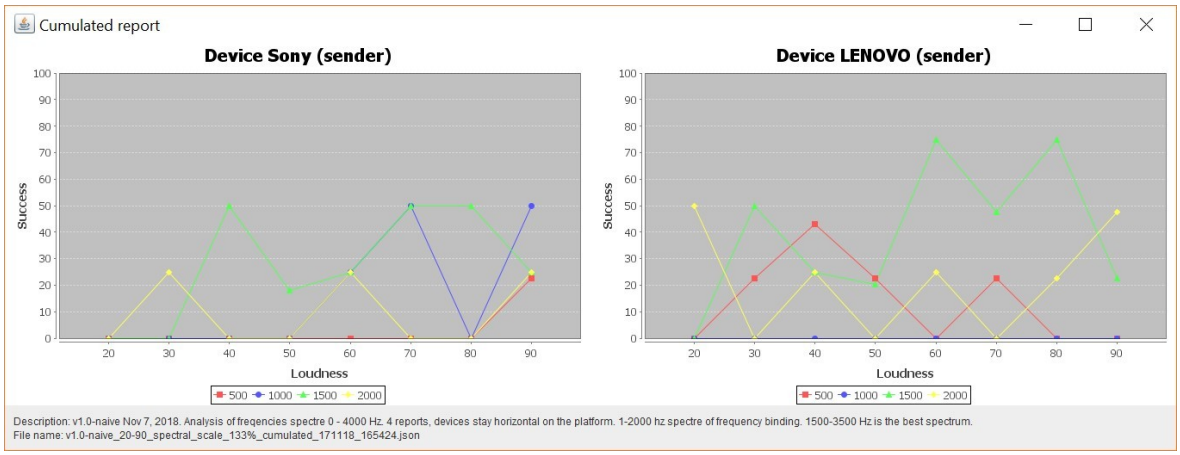
Windows command line. It is needed for automation of configuration of Wi-Fi module in given PC. WindowsUDPServer traps connections of peers connected to the same network.

2.2.3 Elements of cumulated report explained



Img. 23 Example visualization of cumulated report after 2 usual sequential tests

Above screenshot illustrates visualization of a cumulated report which was generated based on test reports of two usual sequential tests. Tests were performed on loudness from 10 to 100. Each line on top charts corresponds to the specific message from a dictionary. Shapes of lines depict dynamic change of success level on different loudness levels. Bottom charts summarize top charts. Values for red line on the bottom chart are calculated using weighted average, where weight is represented by length (in number of characters) of a given message from a dictionary. Blue line represents variance of weighted averages for each separate sequential test report from aggregated weighted average. Can be only < 50. The higher weighted average and lover variance are, the better is total success level for given parameters. During tests illustrated by the above screenshot, the highest success level of communication sessions occurred in 1500-3500 Hz spectrum. For Sony device the best success levels were gained on 60% and 80% of loudness. For Lenovo device the highest success happened on 80% of loudness.

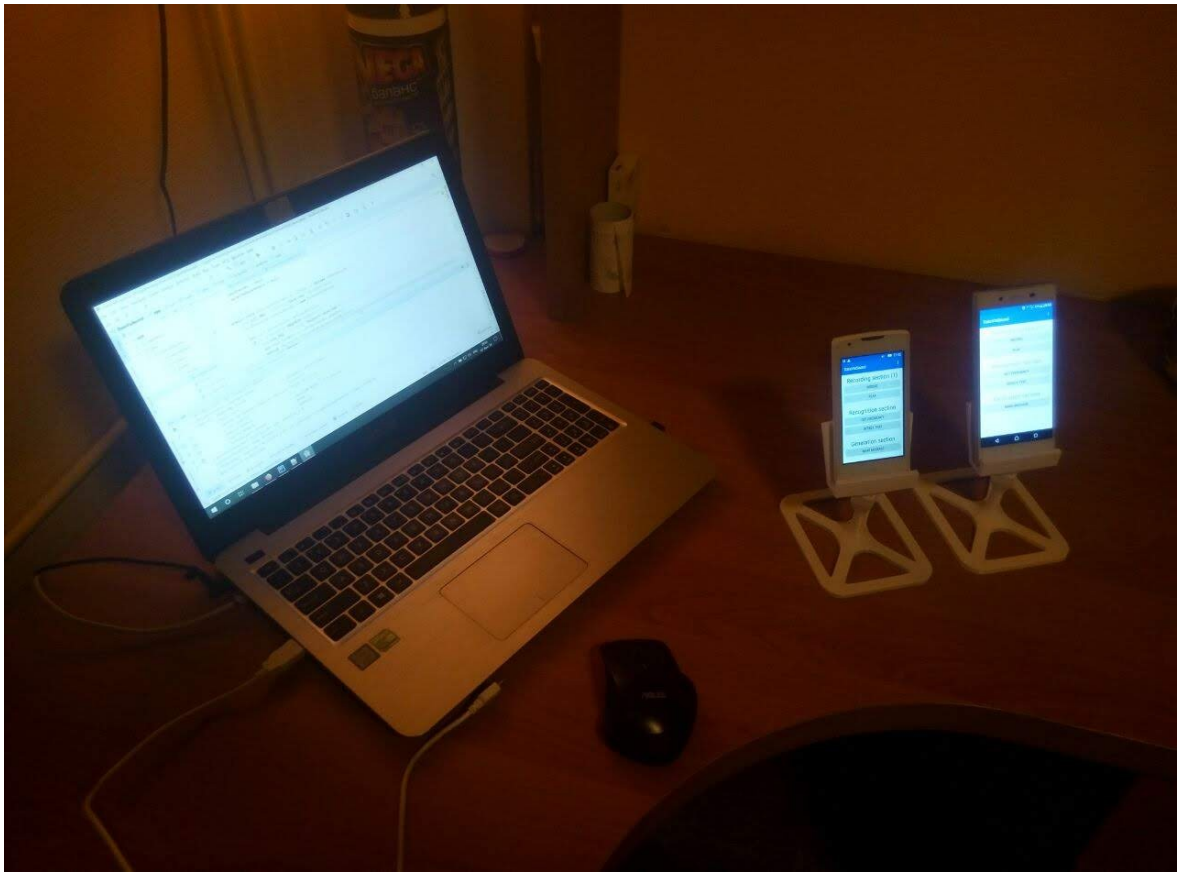


Img. 24 Example visualization of cumulated report after 4 sequential tests with spectral analysis option

Above screenshot illustrates visualization of a cumulated report which was generated based on test reports of four sequential tests with selected spectral analysis option. Frequencies shifts are: 500, 1000, 1500, 2000 Hz. Frequency binding has a range 1-2000 Hz. So it gives us analysis of frequencies in range from 0 to 4000 Hz. During tests illustrated by the screenshot above, the highest success level of communication sessions occurred in 1500-3500 Hz spectrum. For Sony device the maximum success level (50%) was gained on 40%, 70% and 80% of loudness. For Lenovo device the highest success (75%) happened on 60% and 80% of loudness.

Analysis of performed tests comes further in subsection 2.2.5.

2.2.4 Methodology of testing



Img. 25 Workspace when running automated tests

In order to perform tests using DataViaSoundTester, there are some conditions which must be fulfilled:

- Two Android devices and Windows PC with WiFi module, which are located in the same room.
- Android devices must have DataViaSound application installed.
- Desired condition: during all tests, each relevant circumstance should stay the same, except those which are changed for a purpose of test itself.
    - Test operator should ensure that the noise level of surrounding, which is the most important among relevant circumstances, is on the same level during all tests (except if the purpose of the test is to check application work in conditions of constant noise).
    - Devices should be the same during all tests (at least model), except when performing separate group of tests on different devices. While use of two

devices during all tests can give a lot of valuable information for analysis (such as the best frequency, loudness level, overall success level of an algorithm, success level for a particular piece of data), there is an interesting question which can be analysed only with involvement of a third device.

- The question is: is particular success level affected by loudspeaker quality in a sending device or microphone quality in a receiving device. Answer to this question can be approached this way: Let's say we have devices 1, 2 and 3. Then we perform 3 groups of tests: tests between device 1 and 2, 1 and 3, 2 and 3. By comparing results of tests between device 1 and 3, 2 and 3, we can make conclusion about hardware characteristics of devices 1 and 2.

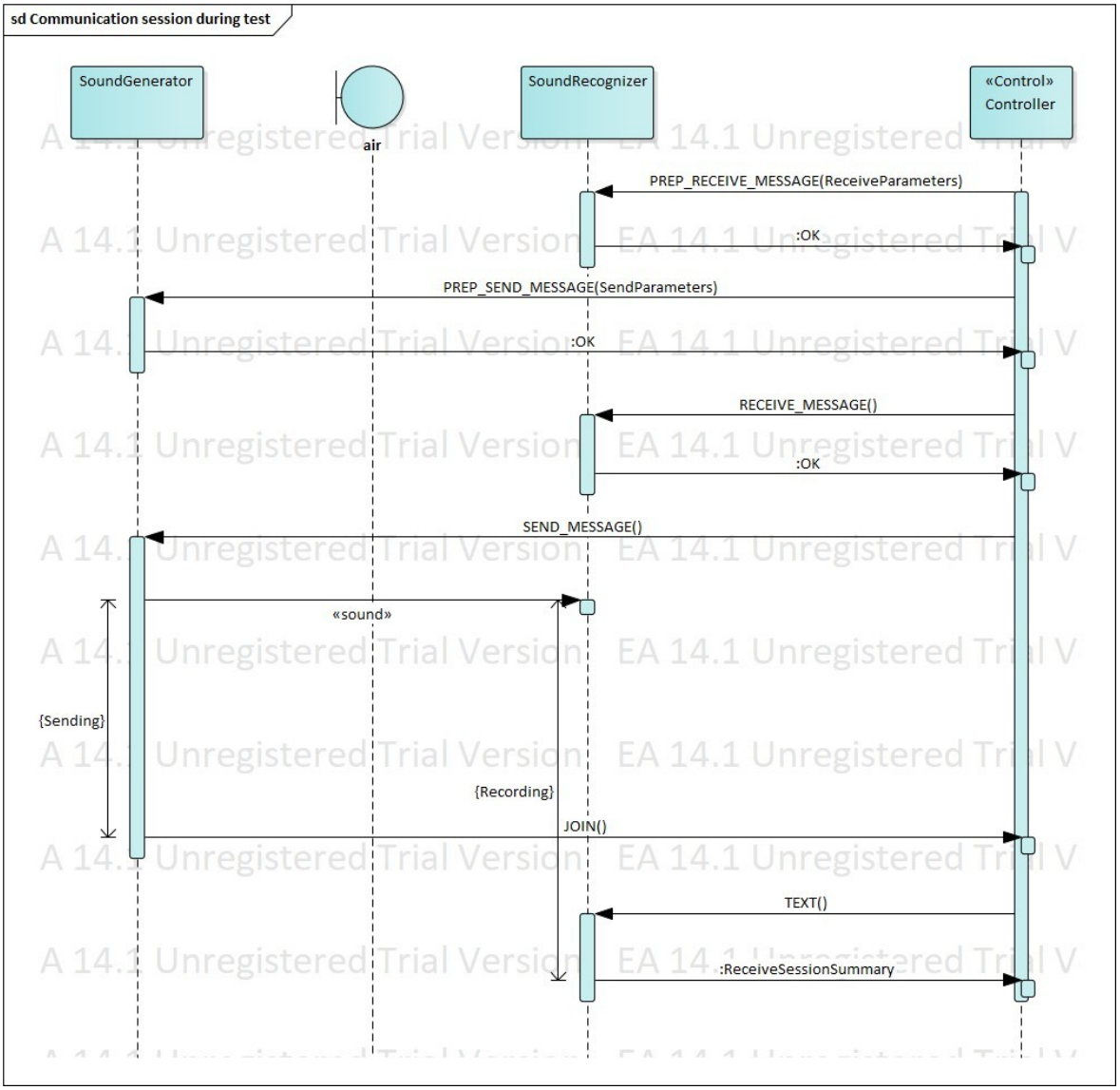
- Distance between devices and their orientation are also relevant circumstances during a test. The key criterion here is how accessible is sender's loudspeaker for receiver's microphone. For the purpose of tests validity, tests were always performed on dedicated platforms in horizontal orientation (so neither microphones nor loudspeakers are covered with any obstacles).

Img. 26 Positions of tested devices

- There is an unstudied possibility that the battery level can change characteristics of produced sound, which can result in different success levels of the communication sessions.

- Cumulated report should be generated with enough sequential tests reports. It is done in order to ensure that resulting cumulated statistical data is averaged enough to be representable.

2.2.4.1 Design of the tests



Img. 27 Activity diagram of communication session between a sender and receiver during tests

In the process of creating DataViaSound, author realized, that planned testing tool has to run hundreds of communication sessions in order to gather relevant statistical data. It means that involvement of test operator should be minimized. So author decided to create testing tool which would be based on client-server model, where sending/receiving devices

are clients (peers) and steering Windows PC would be server (controller). WiFi network was chosen as medium for exchanging the data between a peer and controller.

To run a test program, operator has to complete next steps:

- Create WiFi hotspot with predefined SSID (Heh_mobile).
- DataViaSound has to be running on each peer, menu item “Join tests” has to be clicked.
- Run DataViaSoundTester project with “--runTest” command line argument and select preferred sequential test profile.

DataViaSoundTester then will prepare for running sequence of tests:

- Connect to WiFi hotspot using predefined SSID.
- Trap connections of two peers which are already connected to this hotspot using broadcast via UDP protocol.

During tests, peers accept and execute commands which are sent from the controller. Receiving peer also sends decoded message to controller, where it is used for generating statistical data about performed tests.

Before sending peer starts and after it stops playing a message, there is a half-second pause. It is done to give receiver possibility to make an audio record which would contain irrelevant sound before and after an encoded message (see Img. 5), so walking algorithm can demonstrate it’s effectiveness.

2.2.4.2 Desired improvements of testing environment

The author lived in the dormitory during writing of this work. Thus, this fact can have an impact on results of tests. Some of the environment variables which can influence results of the tests:

- One room for all activities. Cooking, studying, leisure time, guests – everything in one room. An idle human also produces noises – by breathing, coughing, typing on the keyboard, etc. Author’s fridge is very loud, turns on and off periodically. All those factors could influence results of the tests. Separate room for tests would be nice.
- Roommate.

- As in previous point – idle human produces noise. But two people in one room produce even more noise.
- Running more test sequences for one condition would give more averaged statistical data, then results of the tests would be more reliable. Author couldn't afford to run as much tests as he wanted – it would be unethical to his roommate. Testing process is very loud. Sequence of four tests usually takes an hour. 2-3 sequences of tests are usually performed during one day. Still, the roommate was kind enough to be silent during testing process – to make results of tests more clean. It gives hours of listening to loud noise, without possibility to talk freely – surely annoying.

- Devices dedicated specially for tests would be nice. Sony phone was actively used by author. During different tests, charge level of this phone could vary. There could be other unknown states of a phone which could influence results of the tests. For the sake of clean testing, it would be nice if both devices were inactive all the time except testing process, and they were always 100% charged.
- It is possible that surrounding obstacles can influence results of the tests. Reflection of the sound from surrounding obstacles slightly changes characteristics of received sound. Testing workspace was located on author's table – presence and absence of author's body could change results of the tests.

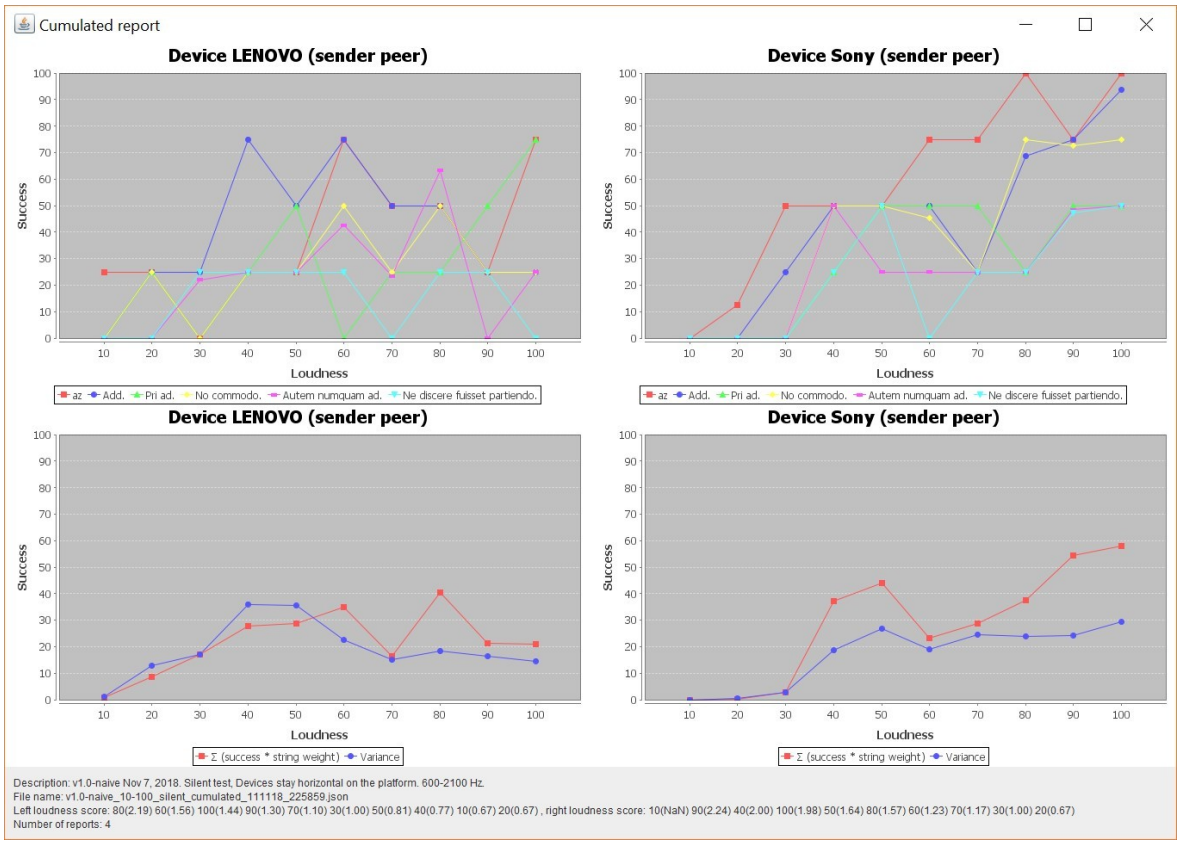
2.2.5 Analysis of existing cumulated test reports

The following subsection contains description of series of performed tests and initial conclusions that results from them. The author deliberately placed tests from various stages of the development of the application and various configurations (including configurations of the test environment). This approach allows capturing more nuances of the obtained results and analyse those results more deeply.

Below list contains only cumulated test reports based on results of sequential tests. Only one sentence "No commodo." was sent during spectrum analysis tests.

2.2.5.1 November 11, 2018

Non-spectral tests, composed of 4 sequential test reports, silent surrounding, beeps were generated on 600-2100 Hz frequencies binding, each beep holds 100 Hz spectrum.

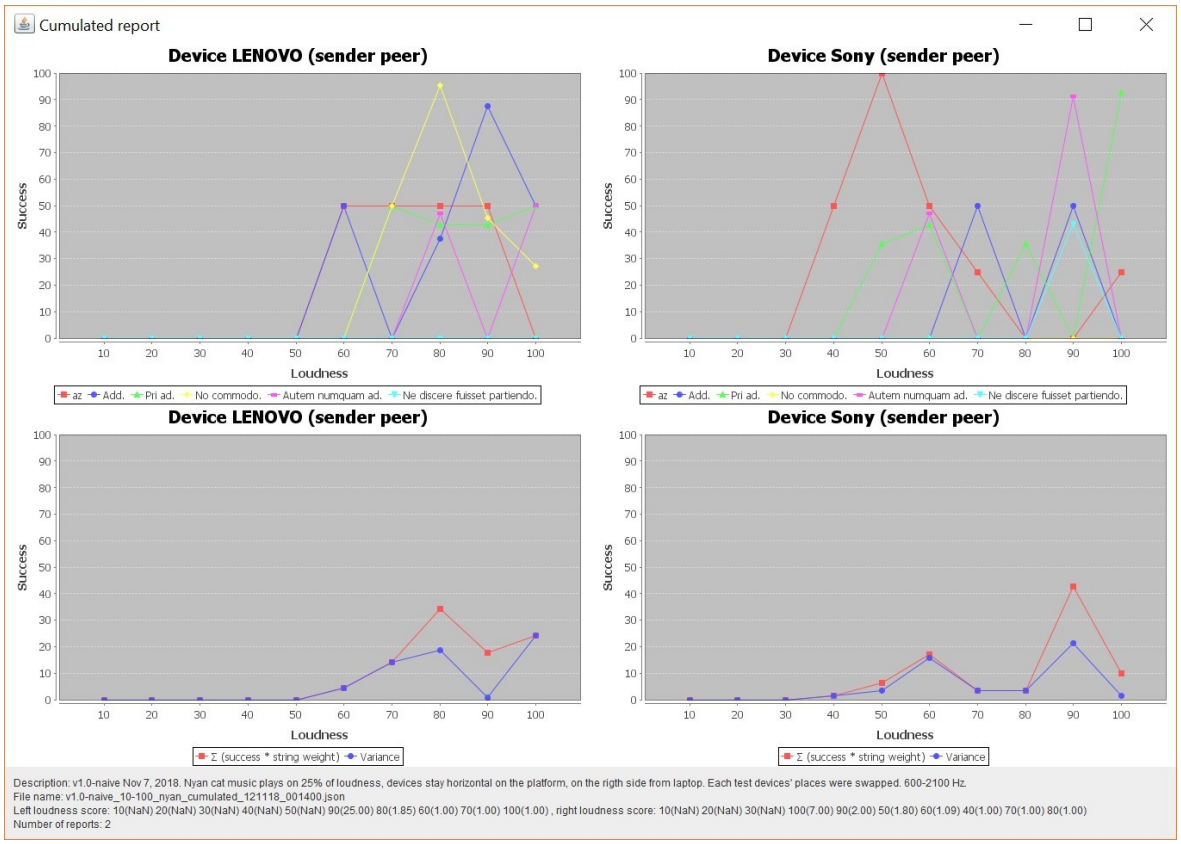


Img. 28 Cumulated report from November 11, 2018

Above report shows that communication sessions usually don't end successfully in current implementation of DataViaSound. The maximum weighted (bottom chart) average of success for Sony is 57% on loudness level 100 and 40% on loudness level 80. The top chart shows us that sending of shorter messages end up better than longer ones. That's why bottom chart considers lengths of messages as a weight for counting average success level for all messages.

2.2.5.2 November 12, 2018

Non-spectral tests, composed of 2 sequential test reports, Nyan cat melody (10 hours edition) runs in the background on 25% loudness of author's laptop, beeps were generated on 600-2100 Hz frequencies binding, each beep holds 100 Hz spectrum.

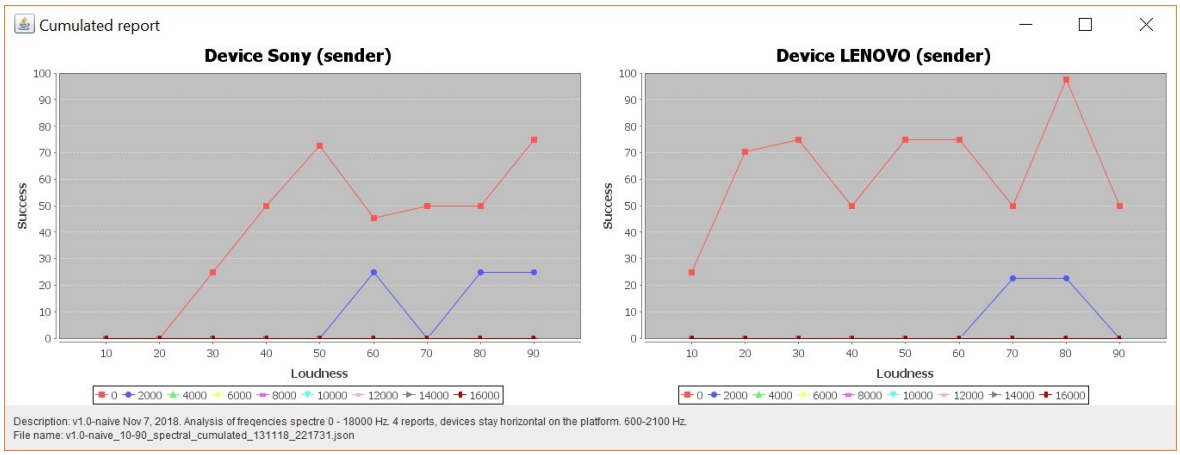


Img. 29 Cumulated report from November 12, 2018, with noise

Since sound is used for communication in DataViaSound, the success levels of communication sessions are worse when there is a background noise. In case of the above cumulated report, the noise was the Nyan cat melody. It was chosen because of its relatively monotonous, short pattern in a loop. So there is no risk that one communication session performs in easier circumstances than another session. Despite success level is still small, we can see a pattern which is supported by the report from November 11, 2018: for 80 for Lenovo and 90-100 for Sony are the best loudness levels.

2.2.5.3 November 13, 2018

Spectral tests, composed of 4 sequential test reports. A total of 9 tests on 600-2100 Hz frequencies binding, with incremental step of 2000 Hz. It gives us analysis of spectrum from 600 to 18100 Hz.

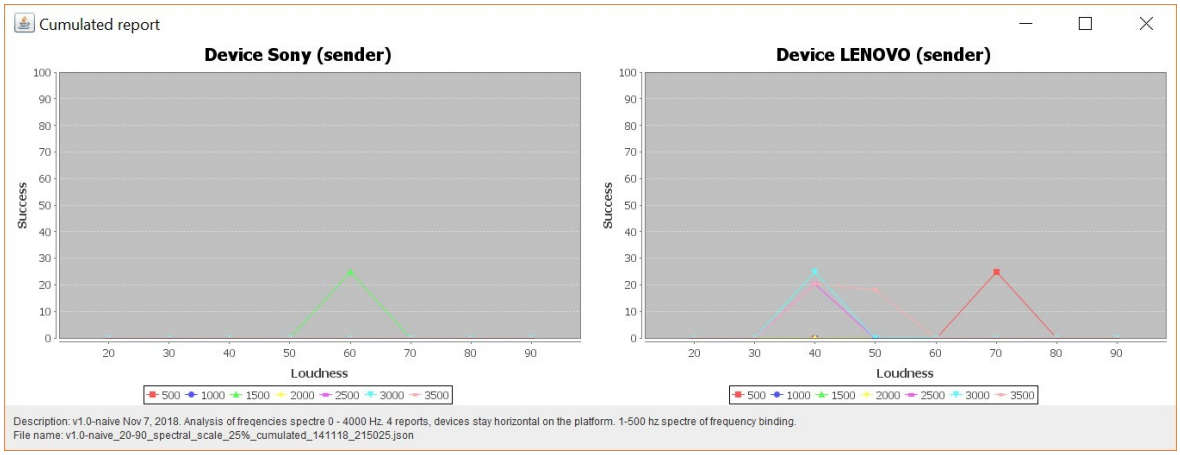


Img. 30 Cumulated report from November 13, 2018

During implementation of DataViaSound v0.1-naive, author intuitively, using the method of tries and faults, decided, that the best frequencies range for communication would be 600-2100 Hz. So this range was used for frequencies binding. But author's choice had to be confirmed or proved wrong by statistical data. Above report shows that author was right in his intuitive choice, frequencies range 600-2100 Hz indeed shows the best results of data transmission. 2600-4100 Hz range demonstrates some success, but much less than 600-2100 Hz range.

2.2.5.4 November 14, 2018

Spectral tests, composed of 4 sequential test reports. A total of 7 tests on 1-500 Hz frequencies binding, with incremental step of 500 Hz. It gives us analysis of spectrum from 500 to 4000 Hz. This test setup was inspired by the report from November 13, 2018, which showed that successful communication is possible in frequencies range from 600 to 4100 Hz.

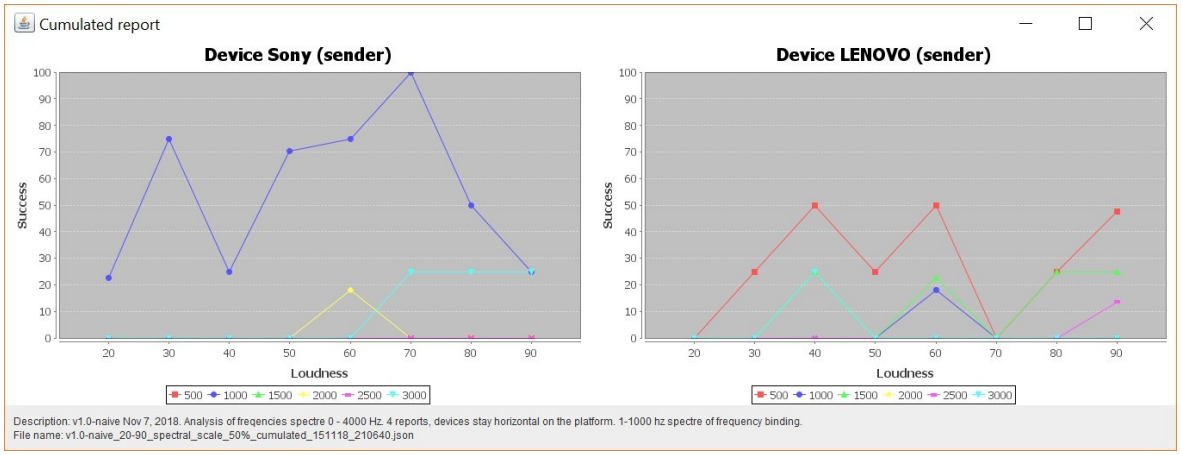


Img. 31 Cumulated report from November 14, 2018

Author suspects that there was an unknown condition during sequence of tests, which caused much worse results (max. 25% of success means that there was only one successful communication out of 4 sequential tests). But most probable reason of such low results is that 1-500 Hz spectrum is too narrow for successful communication.

2.2.5.5 November 15, 2018, with 1-1000 Hz range

Spectral tests, composed of 4 sequential test reports. A total of 6 tests on 1-1000 Hz frequencies binding, with incremental step of 500 Hz. It gives us analysis of spectrum from 500 to 4000 Hz. This test setup was inspired by the report from November 14, 2018, which showed that success level of communication was too low for 1-500 Hz frequencies binding.



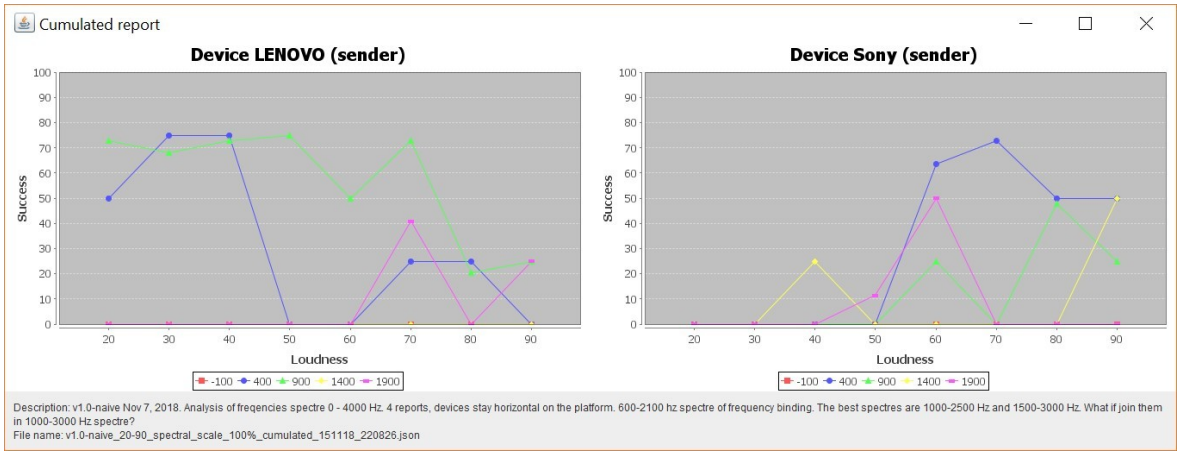
Img. 32 Cumulated report from November 15, 2018, with 1-1000 Hz range

On contrast with report from November 14, 2018, the results are generally better. It can be explained by wider frequency binding range (1-1000 Hz in this sequence of tests instead of 1-500 Hz in previous). Still above report shows some controversy: for Sony 1000-2000 Hz range was the best, but for Lenovo it was 500-1500 Hz range. It would mean that frequencies spectrum would need to be calibrated for each installation of DataViaSound. It seems like a bad idea, since calibration process would take a long time. Which would result in worse usability of DataViaSound. So there is a need to find a universal frequencies range which would more or less suit both tested devices.

2.2.5.6 November 15, 2018, with 600-2100 Hz range

Spectral tests, composed of 4 sequential test reports. A total of 5 tests on 600-2100 Hz frequencies binding, with incremental step of 500 Hz. It gives us analysis of spectrum from

500 to 4000 Hz. This test setup was created with intention to continue search of optimal frequencies range within 500-4000 Hz spectrum.

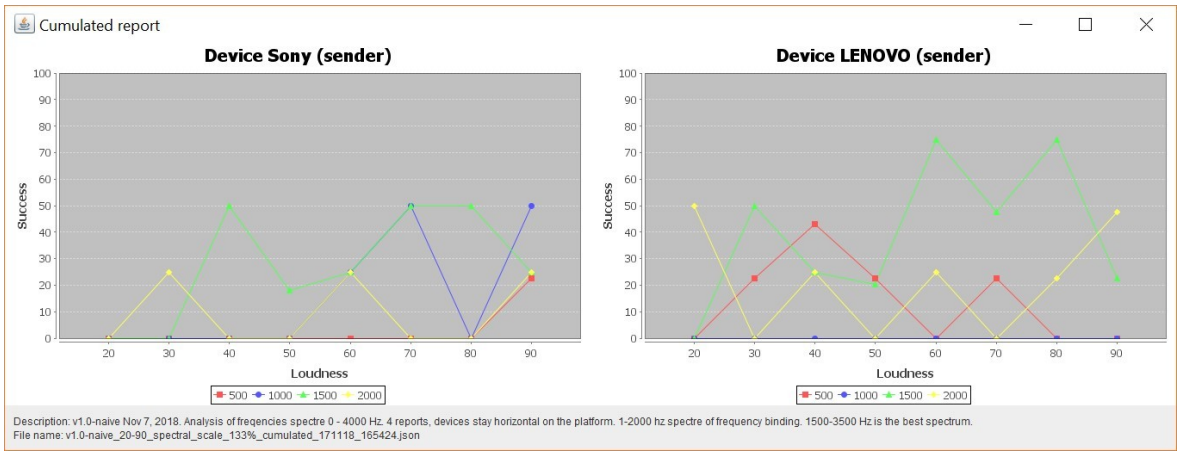


Img. 33 Cumulated report from November 15, 2018, with 600-2100 Hz range

Above report demonstrates that using 600-2100 Hz range it is more possible to find a frequencies spectrum which would suit both devices. 1000-2500 Hz and 1500-3000 Hz ranges demonstrate relatively high success levels for both devices. It is expected that a combination of those two favorite ranges (1000-3000 Hz spectrum) would give good results in next sequence of tests.

2.2.5.7 November 17, 2018

Spectral tests, composed of 4 sequential test reports. A total of 4 tests on 1-2000 Hz frequencies binding, with incremental step of 500 Hz. It gives us analysis of spectrum from 500 to 4000 Hz. This test setup was created with intention to continue search of optimal frequencies range within 500-4000 Hz spectrum.

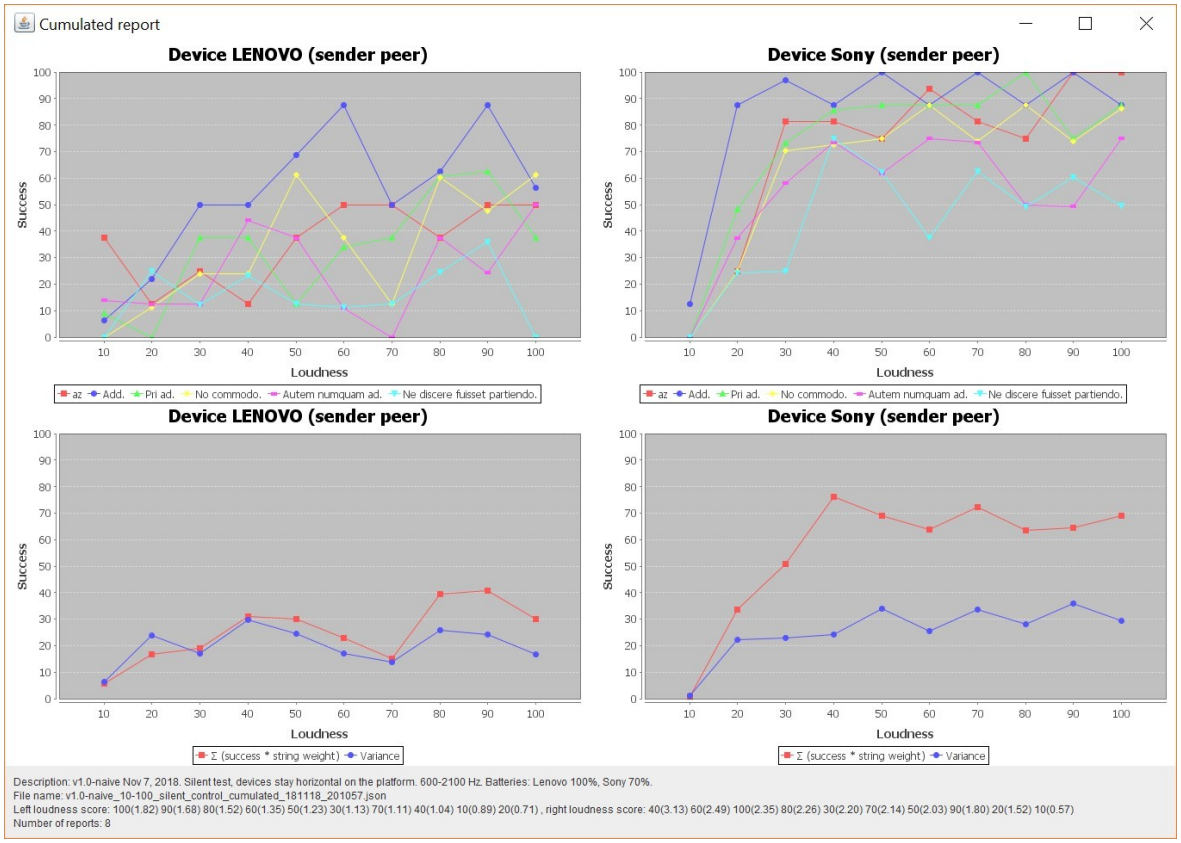


Img. 34 Cumulated report from November 17, 2018

There was an expectation inspired by the Img. 33 Cumulated report from November 15, 2018, with 600-2100 Hz range, that 1000-3000 Hz spectrum would demonstrate the best results. Above report proved it false. Instead, 1500-3500 Hz range turned out to be the best spectrum on both devices. This range demonstrates high results on 60-80 loudness levels on both devices.

2.2.5.8 November 18, 2018 (control test)

Non-spectral tests. Custom attribute of this control test: Frequency spectrum is 600-2100 Hz. See list of static attributes in subsection 2.2.1.



Img. 35 Cumulated report on control test from November 18, 2018

Tests were performed under the same conditions as on November 11, 2018, except there were 8 sequential tests performed this time instead of 4 then. Instead, Sony demonstrated much better results as a sender. Lenovo results are more or less the same. Author can't find an adequate reason for this. Indeterminism of test results appears to be a huge problem. Potential reasons:

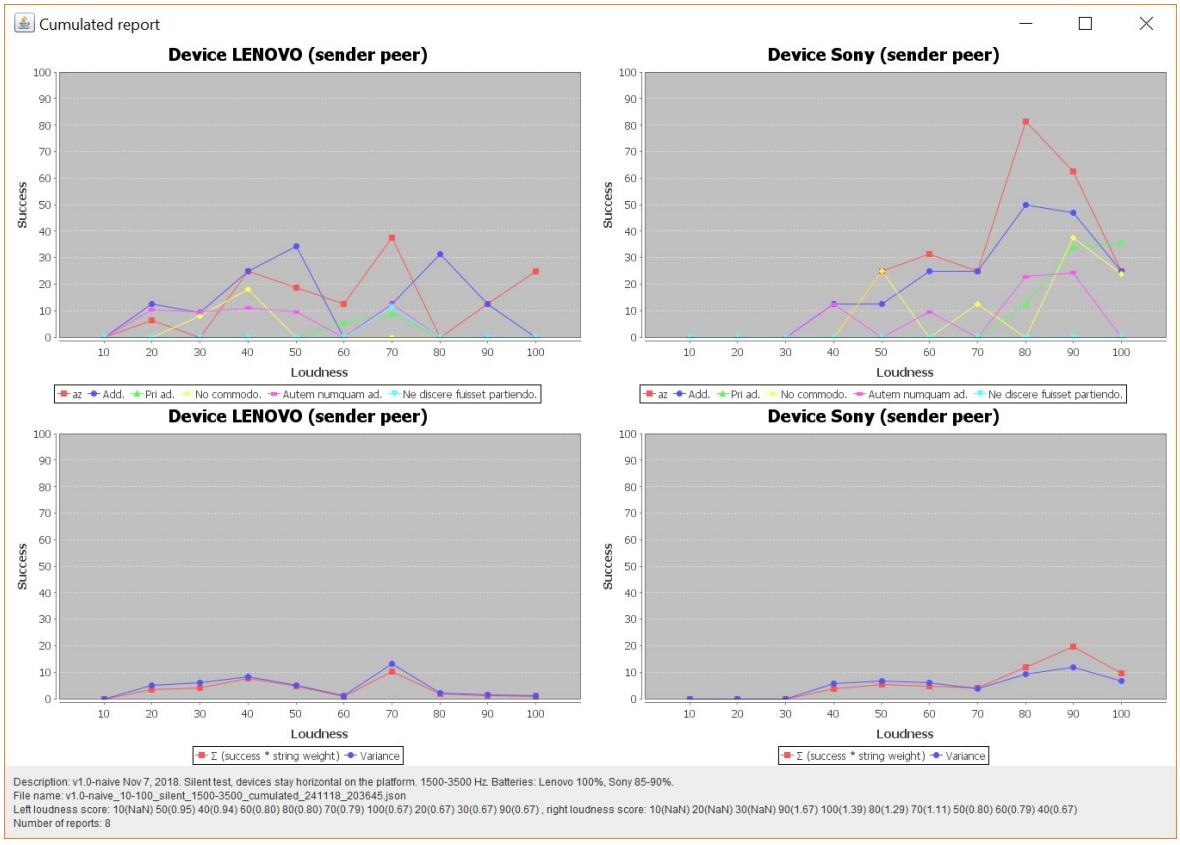
- Author and his roommate weren't present in the room during tests. No small noises of a moving chair, typing keyboard, mouth clicking, breathing, sighing, etc.

- Sony happened to be in a water for significant amount of time recent night. It didn't harm the phone, since it is protected from an impact of water. But it could clean its loudspeaker (not microphone, since Sony wasn't a receiver in those successful tests).

The shape of the variance line almost repeats the shape of average success level for Lenovo as a sender. It means critical indeterminism. While the variance line doesn't completely repeat the average success line in this report, Img. 28 Cumulated report from November 11, 2018 shows that the problem persists also for Sony.

2.2.5.9 November 24, 2018

Non-spectral tests, composed of 8 sequential test reports, beeps were generated on 1500-3500 Hz frequencies binding.



Img. 36 Cumulated report from November 24, 2018

Results of tests summarized by above report show that frequencies spectrum 1500-3500 Hz wasn't proved as better than 600-2100 Hz spectrum. See cumulated report from November 18, 2018 (control test) for comparison.

2.2.5.10 Changes on November 30, 2018

Deeper analysis of error logs in existing tests reports and review of logic in code of DataViaSound has shown two serious problems:

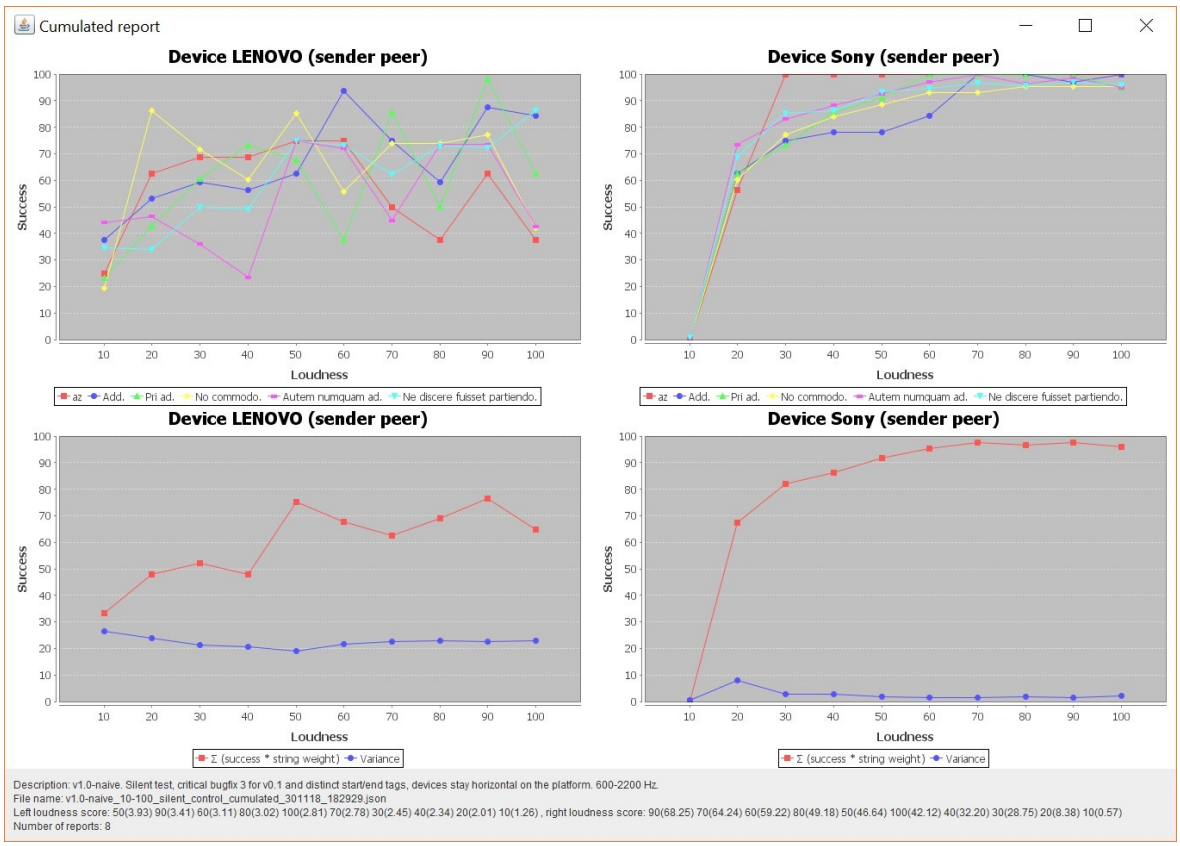
- Logic of exception handling when decoding one particular symbol from hex to char was broken. NumberFormatException with a message like "Invalid int: \"0x\u0000\ u0000\"" wasn't handled in Utils.fromHex(), instead it was propagated to calling class. As result, whole message had been becoming invalid. Instead, in correct implementation, only failed symbol should to be ignored, but the rest of the message should be preserved. This bug made a big share in overall indeterminism of test results, since communication session failed even if one symbol was broken in useful part of encoded message. By design, tests should allow deformation of a received message. This deformation is considered when counting a success level of transformation session. Deformations shouldn't invalidate a received message entirely.
- Exception messages like "length 39; regionStart 2; regionLength -3" appear frequently in test reports. It meant that enclosing tags in encoded messages were too complex. It caused whole decoding process to fail on first step – finding boundaries of a useful message in encoded message – more frequently. Initially, complex enclosing tags were implemented with intention of preventing a situation when decoding algorithm interprets surrounding noise as one of those tags, which would lead to wrong result of decoding (error detection approach). It became obvious, after analysis of some test results, that error detection shouldn't be implemented in this way. Longer enclosing tags mean higher chance that some symbols in those tags will be deformed. Unlike in useful part of an encoded message, integrity of enclosing tags is important for communication session to be valid. If at least one character from enclosing tag is incorrect, then whole message can't be decoded.

Those 2 bugs had huge impact on previous test results, making them invalid. Further tests will be performed with corresponding bugfixes implemented.

2.2.5.11 November 30, 2018 (new control test after Changes on November 30, 2018)

Non-spectral tests. Custom attribute of this control test: Frequency spectrum is 600-2100 Hz. See list of static attributes in subsection 2.2.1. See previous control test from November 18, 2018 (control test) for comparison. Results of previous control test have

become invalid because of the issues described in 2.2.5.10 Changes on November 30, 2018. Changes described there had huge impact on statistics in tests.



Img. 37 Cumulated report on control test from November 30, 2018

See previous Img. 35 Cumulated report on control test from November 18, 2018 for comparison. Results of Sony as a sender improved significantly compared to results of previous control test. High results are reached even on loudness level 30, while variance is on very minimum for all loudness levels. It means stable high result for Sony as a sender and Lenovo as a receiver (Logical conclusion: Sony has a good loudspeaker and Lenovo has a good microphone). For Lenovo as a sender situation doesn't look as good as in case of Sony, which can mean that either Lenovo has low quality loudspeakers and/or Sony has low quality microphone. Still, even for Lenovo situation got better on contrast with previous control test. In the previous report, variance was almost equal to average success level, which meant total indeterminism of results. In current report, average success level is much higher, variance is constant and much lower than average success level.

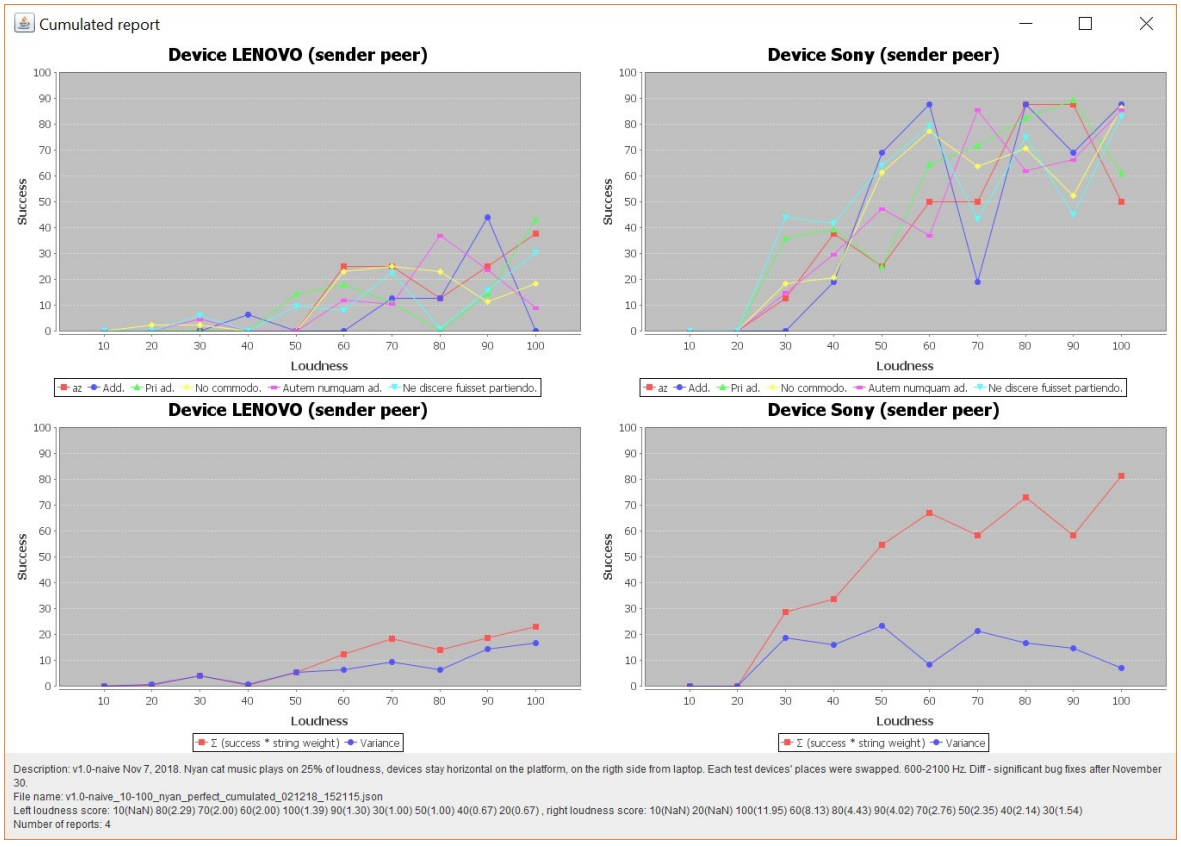
Such improvement of overall success level is a result of Changes on November 30, 2018.

For Sony as a sender, starting with success level of 70% on loudness level 20, the success level is slowly growing, until it reaches the success level 95% on loudness level 60. Use of higher loudness levels demonstrates success levels above 95%.

For Lenovo as a sender, top success level is 75%, which happened on loudness levels 50 and 90.

2.2.5.12 December 2, 2018

Non-spectral tests, composed of 4 sequential test reports, Nyan cat melody (10 hours edition) runs in the background on 25% loudness of author's laptop. See Img. 29 Cumulated report from November 12, 2018, with noise for a previous version of this test. This test was repeated because of the Changes on November 30, 2018.



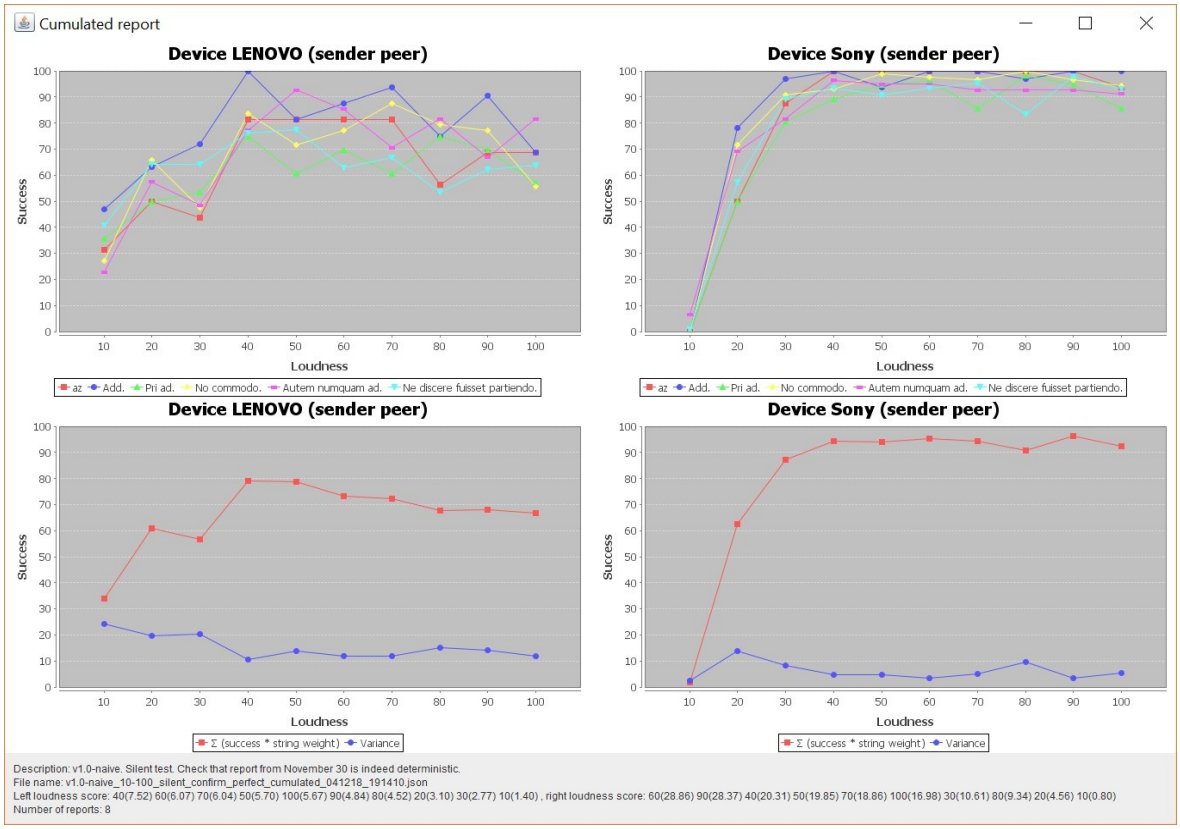
Img. 38 Cumulated report from December 2, 2018

Again, significant improvement of results for Sony → Lenovo relation. Relatively high results in this configuration (Sony → Lenovo relation, Nyan cat melody in the background) can be explained by assumption, that loudness-distance relation is more strong in Sony → Lenovo relation, than in laptop → Lenovo relation. Meaning, that, taking into account distance between devices and loudness of both laptop and Sony, Sony wins – has a

stronger, more dominant signal related to noise from the laptop. But in Lenovo → Sony relation, the same noise parameter caused poor results. The reason is eventually the same as for previous lower results of Lenovo → Sony relation: mainly hardware quality, eventually combination of hardware specifics with given operational frequencies binding (some loudspeakers could produce more distorted/clean sound on specific frequencies, also some microphones could record specific frequencies more or less distorted).

2.2.5.13 December 4, 2018

Non-spectral tests, composed of 8 sequential test reports, beeps were generated on 600-2200 Hz frequencies binding. Tests were performed in the same conditions as November 30, 2018 (new control test after Changes on November 30, 2018). Purpose of the repeat: prove that tests indeed had become deterministic after the Changes on November 30, 2018.



Img. 39 Cumulated report from December 4, 2018

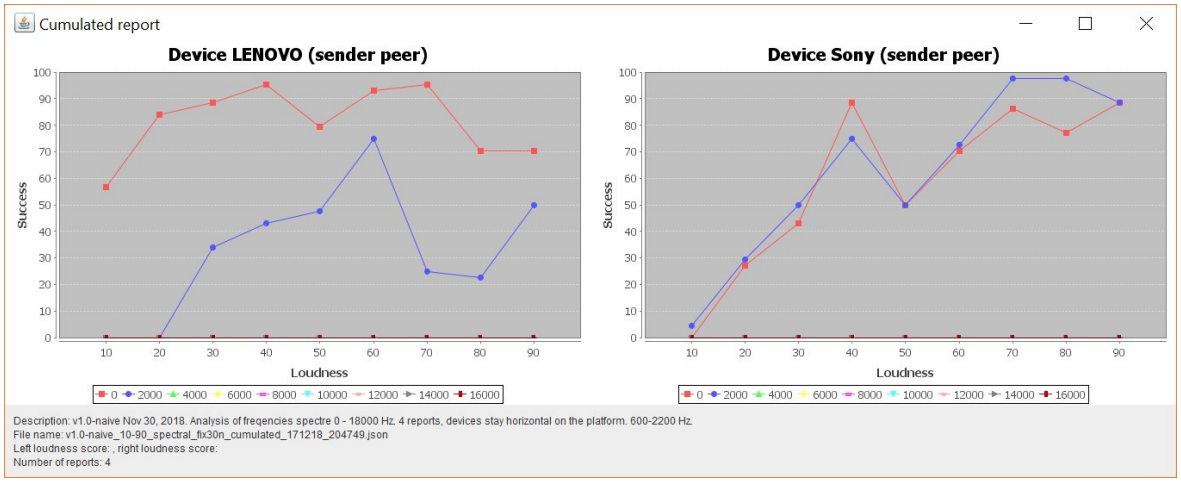
Above report demonstrates that tests had become deterministic after Changes on November 30, 2018. Before those changes, sequences of tests ran under the same conditions showed noticeably different results (compare reports from November 11, 2018 and November 18, 2018 (control test)). In above report, Sony as sender demonstrates almost identical results compared to tests on November 30. Lenovo as sender demonstrates

a little bit different results of average success level than in previous sequence of tests (especially on loudness levels 20 and 40). Still it doesn't mean indeterminism, because:

- the difference is moderate,
- the difference appears on smaller loudness levels (which are more sensitive to random surrounding circumstances),
- variance line:
  - doesn't repeat the shape of the average success level line (versus before the Changes on November 30, 2018),
  - for Sony is 5-15% and for Lenovo 10-25% (versus 25-35% and 15-30% respectively on November 18, 2018 (control test)).

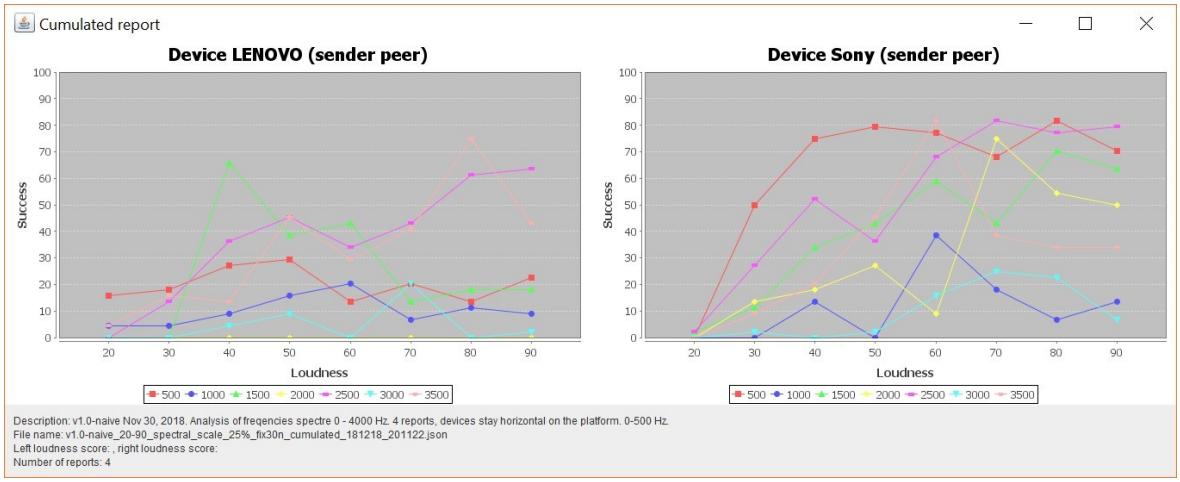
2.2.5.14 December 17 – 18, 2018 (remake of spectral tests after Changes on November 30, 2018)

See spectral tests between November 13, 2018 and November 17, 2018. Purpose of those spectral tests is the same as for previous ones – to find the best range of frequencies in available spectrum from 0 to 18000 Hz. Test parameters are the same – except that Changes on November 30, 2018 had significantly improved results of the tests, thus tests should've been run again. As seen on reports below, results improved indeed compared to those between November 13, 2018 and November 17, 2018 – success levels are sometimes higher even twice.



Img. 40 Cumulated report from December 17, 2018, analysis of frequencies in 0 – 18000 Hz spectrum

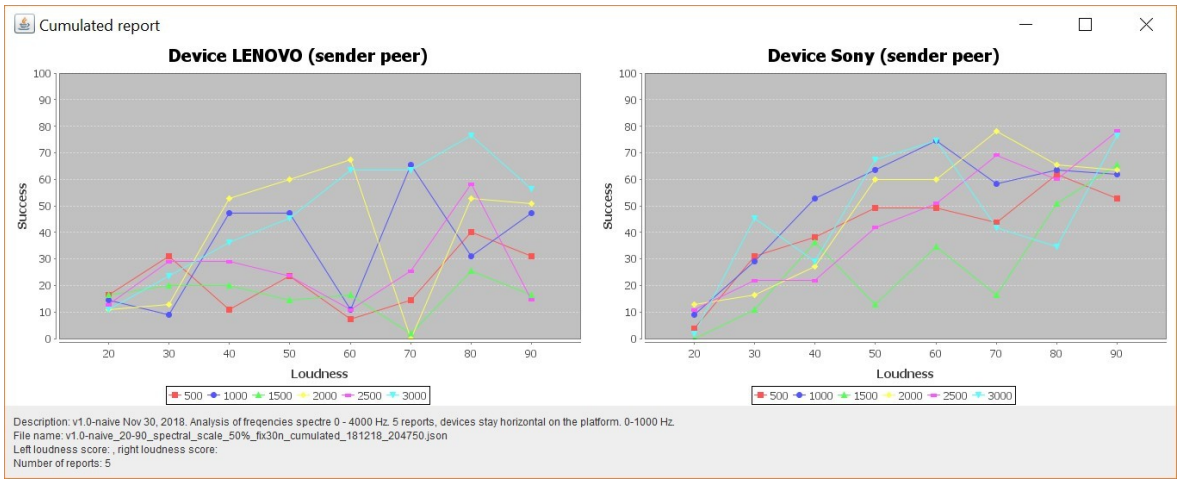
As in the report from November 13, 2018, tests in 0 – 4000 Hz frequency range still shows the best results. It means that more detailed spectrum tests should be performed within this range.



Img. 41 Cumulated report from December 18, 2018, analysis of frequencies in 0 – 4000 Hz spectrum with 0 – 500 Hz frequencies binding

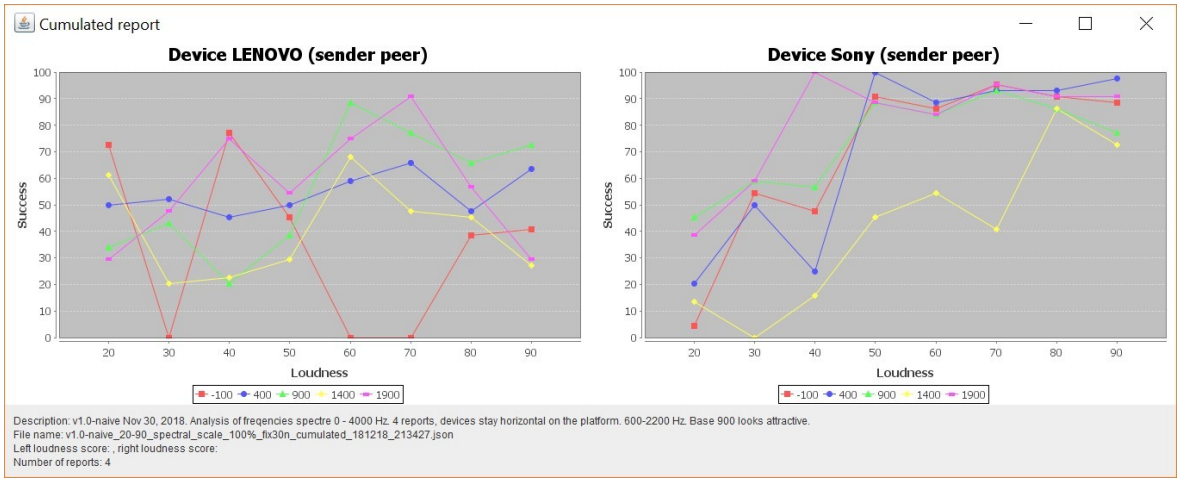
Conclusion from the above cumulated report: 2500 – 3000 Hz range is quite successive on higher loudnesses, it is true for communication in both directions. Other ranges of frequencies also could be high (as 500 – 1000 Hz range), but only for communication in one direction. Choice of frequencies range is good if success level, at least for some loudness levels, is:

- The same for communication in both directions.
- Is close to 100%.



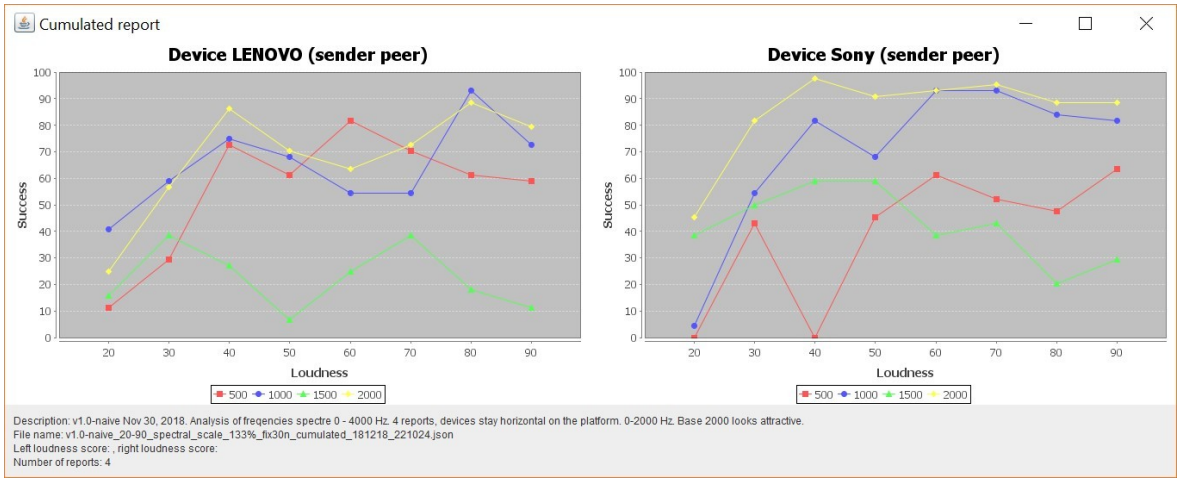
Img. 42 Cumulated report from December 18, 2018, analysis of frequencies in 0 – 4000 Hz spectrum with 0 – 1000 Hz frequencies binding

Assuming above cumulated report, there is no good frequency range to choose. Results of all frequency ranges are poor and none of them demonstrate the same dynamics for communication in both directions.



Img. 43 Cumulated report from December 18, 2018, analysis of frequencies in 0 – 4000 Hz spectrum with 600 – 2200 Hz frequencies binding

The situation demonstrated on above report is better – at least for Sony → Lenovo direction. Tests performed on loudness levels from 50 to 90 finished with 85-95% of success for Sony → Lenovo relation. It allows to choose any frequency for Lenovo → Sony relation that has similar success percentage for same loudness levels. The green line on the left chart tells that 1500 – 3100 Hz is a good candidate for being new (operational) frequency binding for whole Application, as its success varies between 70-90% within 60-90 loudness levels.



Img. 44 Cumulated report from December 18, 2018, analysis of frequencies in 0 – 4000 Hz spectrum with 0 – 2000 Hz frequencies binding

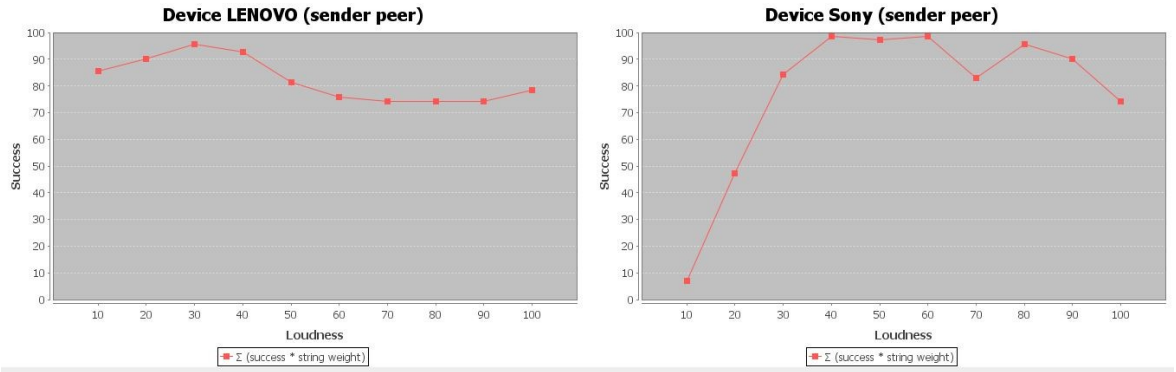
Taking into account criterias described previously, 2000 – 4000 Hz frequency binding is also good candidate for being selected as operational frequency binding in DataViaSound.

Next two non-spectral tests will check success levels of communications for 1500 – 3000 Hz and 2000 – 4000 Hz frequencies bindings. For the best of them, control test will be performed. Its result will be compared to November 30, 2018 (new control test after Changes on November 30, 2018), which utilized 600 – 2200 Hz frequencies binding. If new frequencies binding is better, it will be used as operational for whole DataViaSound application.

2.2.5.15 December 19 – 20, 2018 (comparison of 1500 – 3000 Hz and 2000 – 4000 Hz frequencies bindings, indeterminism again)

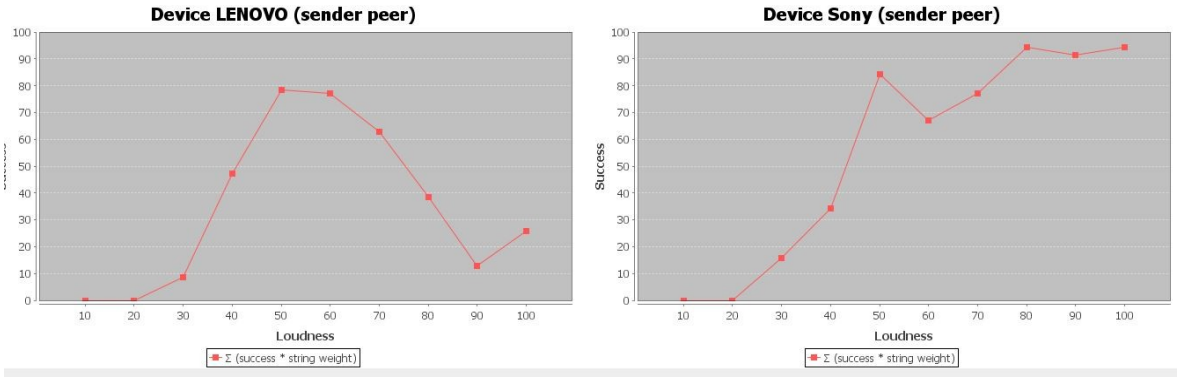
Reports of non-spectral tests for 1500 – 3000 Hz and 2000 – 4000 Hz frequencies bindings are presented below. As stated in subsubsection 2.2.5.14, here you'll also find comparison between those two frequencies bindings and decision on which frequencies binding will be used for next control test.

In the evening of December 19, 2018, after performing comparison tests for mentioned frequencies, author noticed indeterminism of results again – not as dramatic as before Changes on November 30, 2018, but still serious. For 4 tests in total, first two were perfect – success levels were very high, in stable way.



Img. 45 Typical average success level for first two sequential tests

Suddenly, right in the middle of the testing process, results got diametrally different – success level had dropped, many tests had been finishing with 0% success. Author decided to stop the test, restart the devices and finish the last two sequential tests. Results improved slightly, but still didn't return to the previous trend.

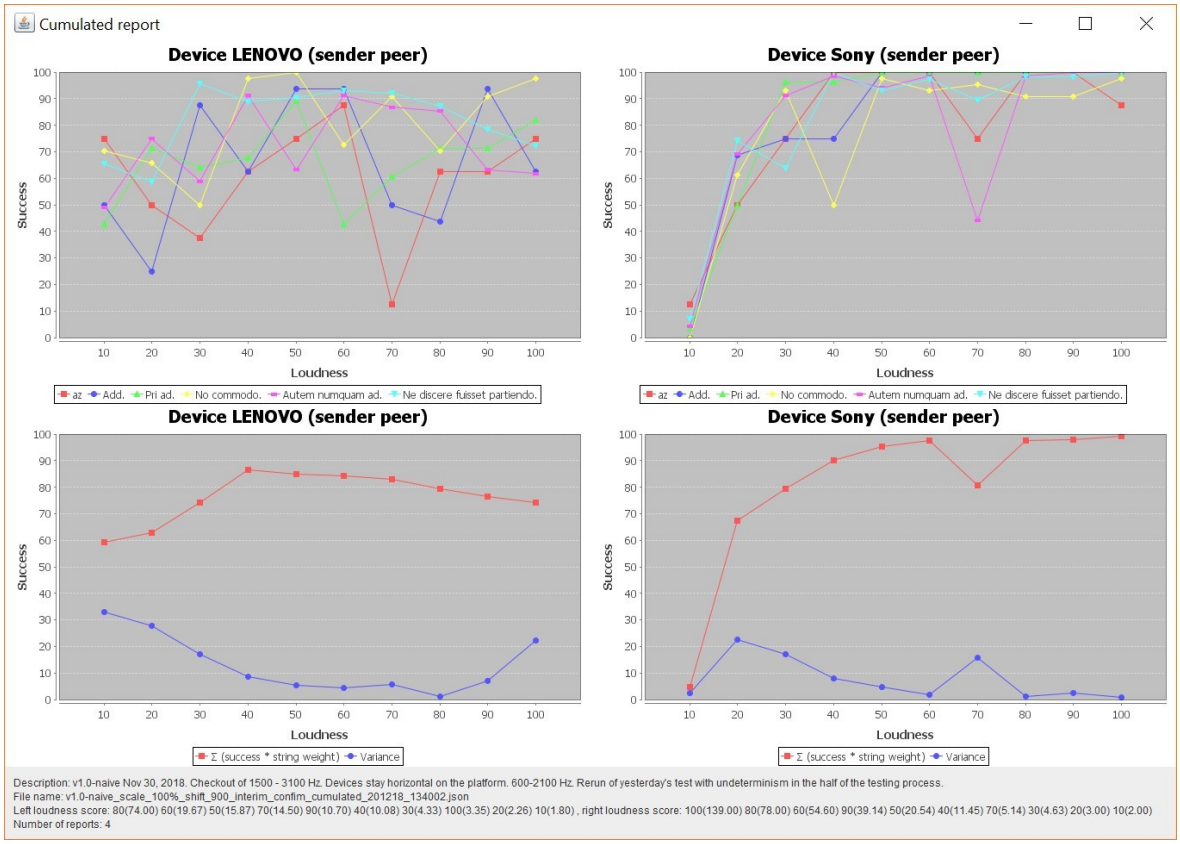


Img. 46 Typical average success level for last two sequential tests

Battery level for Sony was 30-50% that evening, for Lenovo 100%. Still, author's subjective impression – battery level doesn't make significant impact on success level of communication sessions. He had many occasions to compare results of tests for different battery levels, didn't notice any significant difference in results. Example battery levels during comparison: 100% and 30% for Sony, Lenovo battery level was almost always 100%. It could be also, that hardware of the phone was somehow 'exhausted' after a day of work (a case for Sony, which was active for whole day). The problem of how battery level impacts success level of communication session requires more investigation.

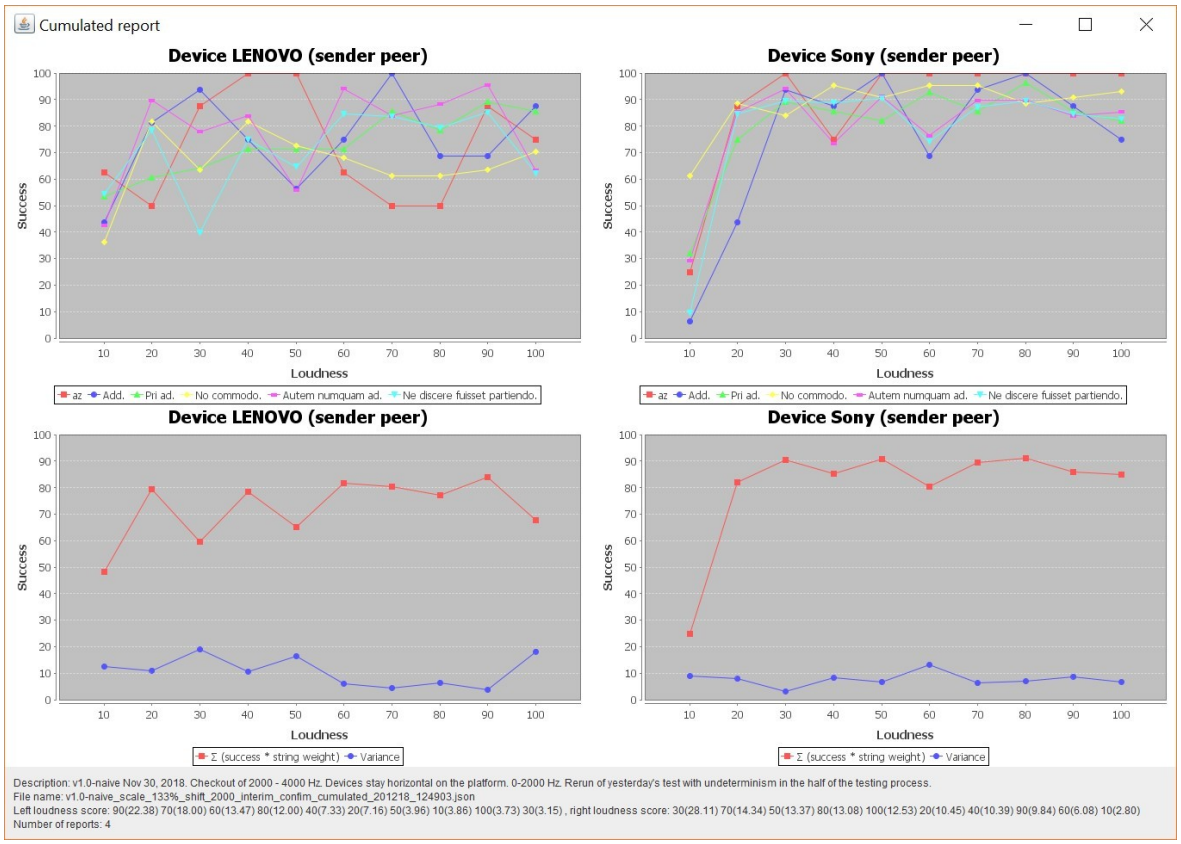
Img. 47 Cumulated report which contains first two good tests and last two indeterministic (after restart)

Author decided to rerun same tests next day, since results of indeterministic tests can't be used for judging if the situation was improved or not.



Img. 48 Cumulated report from December 20, 2018, for 1500 – 3100 Hz frequencies binding

Above report, compared to the same from previous day, demonstrates better results. Variance is settled down, success level is higher. Taking into account background of previous cumulated test reports, success levels are very high in both directions of communication. For relation Lenovo → Sony this success level is surely extraordinary. Even more, besides success level being, it's chart is smooth regarding loudness levels – no drastic 'jumps' of red line. The reason for such good result is that an unknown cause of indeterminism from previous day was gone.



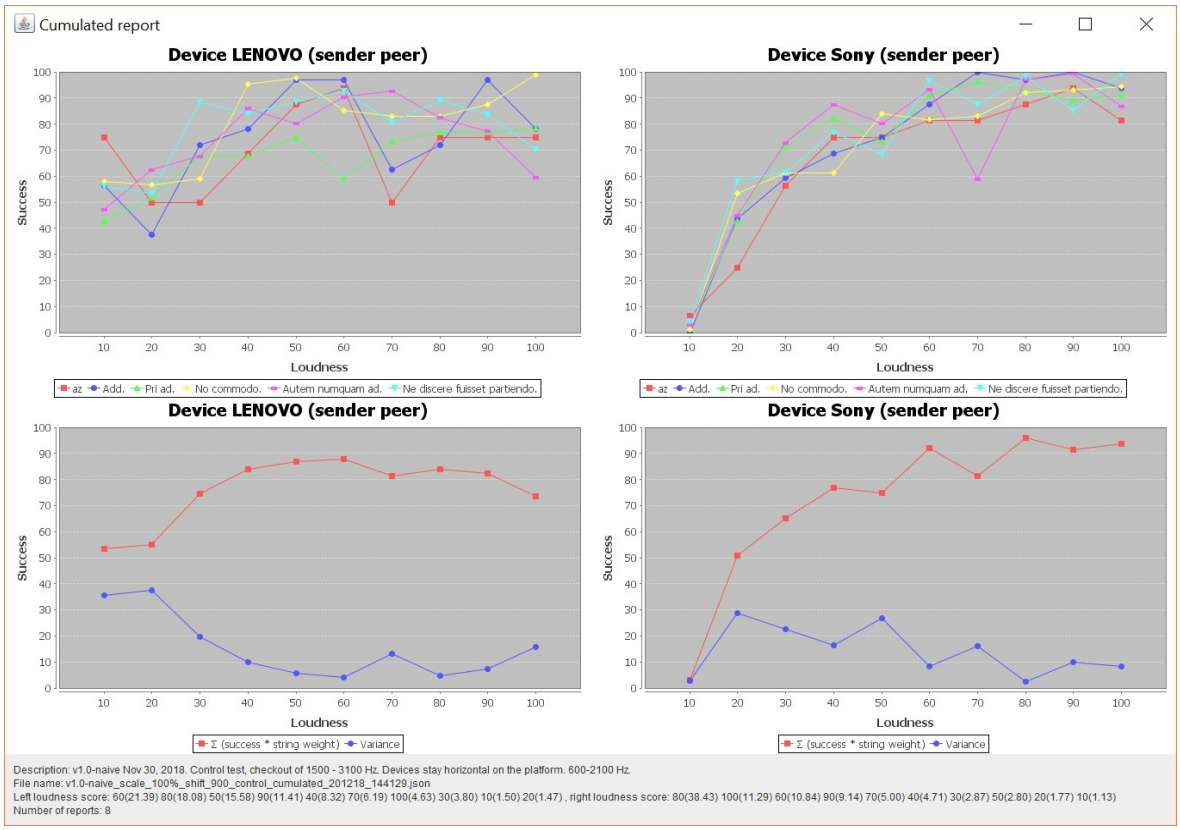
Img. 49 Cumulated report from December 20, 2018, for 2000 – 4000 Hz frequencies binding

Tests ran on 2000 – 4000 Hz frequencies binding demonstrated a little lower (on scale of 5%) success level, results were a bit less predictable (success level and variance lines are not smooth), on the other hand, variance generally is lover throughout whole charts.

With a small transcendence, 1500 – 3000 Hz frequency binding is better. In this configuration, 50 and 60 loudness levels are the most suitable for communication in both directions. Success level is high, variance is low for both Sony → Lenovo and Lenovo → Sony relations. Those facts enabled author to make a decision for performing a control test for 1500 – 3000 Hz frequency binding.

2.2.5.16 December 20, 2018 (control test for 1500 – 3000 Hz frequencies binding)

Non-spectral tests. Custom attribute of this control test: Frequency spectrum is 1500 – 3000 Hz. See list of static attributes in subsection 2.2.1.



Img. 50 Cumulated report from December 20, 2018

See previous Img. 37 Cumulated report on control test from November 30, 2018 (600 – 2100 Hz frequencies binding) for comparison. Success level improved for Lenovo → Sony relation, got worse for Sony → Lenovo relation. Variance got higher for Sony → Lenovo relation, decreased for Lenovo → Sony relation between 40 and 90 loudness levels.

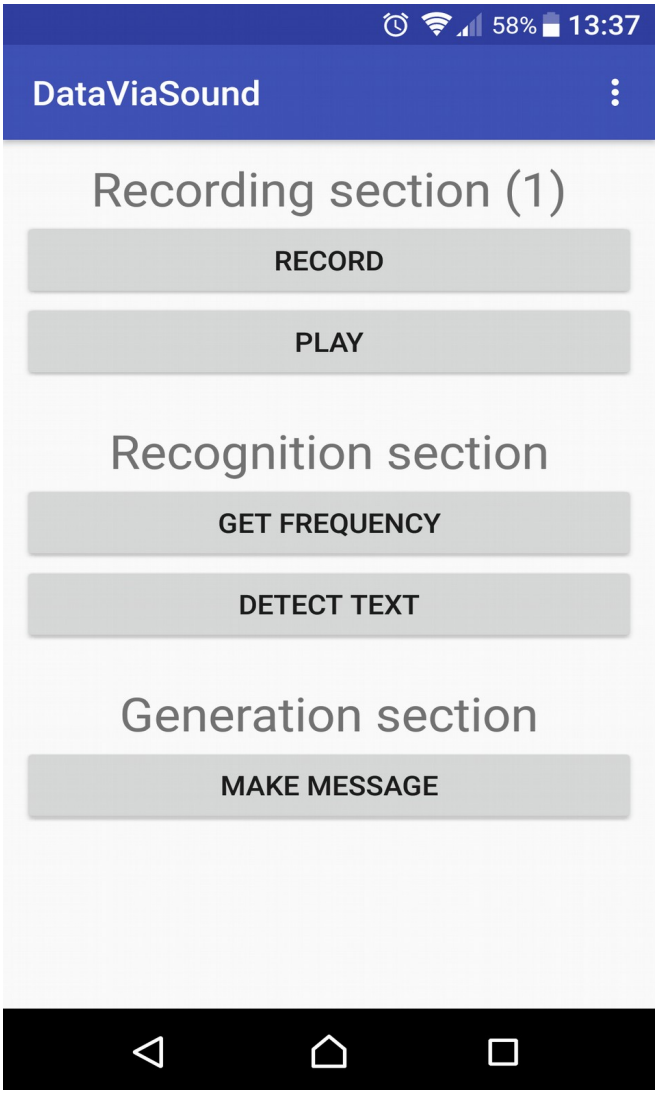
Author had decided that, from now on, frequency spectrum 1500 – 3000 Hz and loudness level 60 are new operational parameters for DataViaSound v1.0.

2.2.6 Known bugs of the testing tool

1. [FIXED] Sometimes ArrayIndexOutOfBoundsException was throws by library PvdBerg1998/PNet. It caused whole sequential test to abort. A tricky thing about this bug was, that the library handled exception by just printing a short message to console, but not printing a stacktrace of an error. So author was assuming for long time that an error happens inside of the library, and that it is the fault of a developer of the library. At some point of time, this exception started to appear constantly. Author decided to investigate deeper. It tuned out that exception actually happened in author's code, during processing of a result of a communication session (SingleTestResult.countMatchPercentageCharByChar()).

2.3 DataViaSound v1.0 – improvements and bugfixes

DataViaSound v1.0 – version of the application, which includes changes made during this diploma project. Bugfixes, improved sending/receiving algorithm, implementation of functionality for automated testing, tweaks in communication parameters (like frequencies binding, loudness level, enclosing tags, etc.).



Img. 51 Screenshot of the main screen of DataViaSound application

Author didn't plan to put an effort into the look and design of the application, since it wasn't related to aims of this work.

Detailed list of bugs and improvements which were introduced in the new version of the application, as a result of this diploma work:

- [IMPLEMENTED] Indicate enclosing tags more explicitly, so that decoding algorithm at least doesn't fail on finding of the start and end of a useful message.

See 2.2.5.10 Changes on November 30, 2018 for detailed description of the problem.

  - Only one beep is used for indication of enclosing tags (instead of 3 beeps previously). It had reduced probability of false decoded characters of the tag, since the tag is shorter itself.
  - Combine this solution with dedicating specific frequency only for carriage beeps. Let's say, now 600-2100 Hz frequencies spectrum is used entirely for encoding the message. New solution adds use of 2200 Hz only for indicating of the enclosing tags. This carriage frequency wouldn't be used for encoding of a useful message. This change would make enclosing tags more explicit on background of other beeps.

- [BUG FIXED] NumberFormatException with a message like "Invalid int: \"0x\u0000\u0000\"" wasn't handled in Utils.fromHex(), instead it was propagated to calling class. It caused whole communication session to fail. See 2.2.5.10 Changes on November 30, 2018 for detailed description of the problem.
- [IMPLEMENTED] Set 1500-3100 Hz frequencies binding and loudness level 60 as operable for usual communication sessions in the application.
- [IMPLEMENTED] Made sure that system loudness level is always maximum during sending of a message. There are two loudness levels which make an impact on actual loudness of sound produced by application in Android: system loudness (a user can regulate) and loudness of current AudioTrack object from Android API. To ensure equal conditions for each communication session, author had decided that system loudness level should be set to the maximum during message sending, but actual loudness would be adjusted using AudioTrack object.

3 Conclusions

An overview about what was done during the work and it's results. Conclusions. Assumptions about future development of this work and next potential steps which can be done in the project.

3.1 DataViaSound v0.1

After v0.1 was checked with testing tool, it was confirmed that success level of communication was low. It was between 30 and 70% (see Img. 35 Cumulated report on control test from November 18, 2018), which means that there was a lot of space for improvement. Such small success level also means that initial implementation was too far from being used for robust transportation of data.

Pro of initial implementation:

- small modification of code allows to transfer any raw byte data using Base64 encoding, since version 0.1 can send/receive ASCII symbols.

Cons of initial implementation:

- High error rate, major reason is described in subsection 2.1.6 Bugs and improvements of version 0.1. Also, errors happen more frequently when surrounding is not silent.
- Small bandwidth of 5 bytes/second.

3.2 Conclusions after the tests

There are some general conclusions which come up after Analysis of existing cumulated test reports (see subsection 2.2.5):

- Overall success level of communication is not satisfactory. It means that current implementation of DataViaSound is not reliable for practical data transportation. But it is a subject to improvements.
- Intuitive choice of frequencies spectrum 600-2100 Hz was surprisingly correct. After shallow testing, this spectrum demonstrates the best success level in 500 - 16000 Hz spectrum (see Cumulated report from November 13, 2018, also further reports up until November 17 were focused on finding the best frequency). More

spectral tests are needed though, there is a possibility to find slightly better spectrum within 500-4000 Hz range. After Changes on November 30, 2018, this conclusion might be no longer valid, so tests should be ran again in the same conditions (which author didn't do because of lack of time).

- Most of the fails in tests happen because a start/end of a message can't be detected. Enclosing tags are apparently hard to detect. Repaired by Changes on November 30, 2018.
- Exception handling during hex-to-char conversion wasn't done correctly. Some hex numbers could be deformed during communication session, so they are not suitable for hex-to-char conversion. NumberFormatException with a message like "Invalid int: \"0x\u0000\u0000\"" wasn't handled in Utils.fromHex(), instead it was propagated to calling class. It led to whole communication session to fail. Instead, a broken symbol should be ignored, but the rest of hex sequence should be processed as usual. Majority of exceptions during decoding happen because of this issue. Repaired by Changes on November 30, 2018.
- Diagram for a particular device doesn't evaluate only characteristics of this device as a sender, but also of the second device as a receiver. High success level on a diagram means good quality microphone in a receiver and loudspeaker in sender. Low success level, on another hand, can mean a bad quality microphone in a receiver or/and loudspeaker in sender. The situation when we have low success level is worse, since there are three possible causes of the problem: a bad quality microphone in a receiver, a bad quality loudspeaker in a sender and both.

3.3 DataViaSound v1.0

Generally, the situation was significantly improved compared to v0.1. During development of this work, there were some critical bugs found in the code, which caused dramatical drop of test results (bugs were fixed, see 2.2.5.10 Changes on November 30, 2018). After mentioned bugfixes, success level for standard application configuration was stable for most of the loudness levels between 80 and 100% (see Img. 37 Cumulated report on control test from November 30, 2018). Also, search for the best frequencies binding turned out to be successful. Application which operated on frequencies binding 1500 – 3100 Hz demonstrated stable and relatively equal success level for Sony → Lenovo and Lenovo → Sony relations above 70% (see Img. 50 Cumulated report from December 20,

2018), which is better result compared to previous control test, since results are equally predictable for communication relations in both direction. Loudness level 60 was selected as operational, because, based on the most recent control test, results were the highest for relations in both directions.

3.4 General conclusions

Author has succeeded in achieving better results of communication sessions in DataViaSound. DataViaSoundTester tool was very helpful in the overall process of improvement and development of DataViaSound. It enabled author to track success level for different implementations of the application, compare success levels for different operational frequency ranges and note optimal loudness levels.

3.4.1 Potential improvements in future versions of DataViaSound

The topic of this diploma work turned out to be very complex and needs more studies. In perspective, below unresolved questions can be addressed in Master diploma work:

- Use an error detection/correction algorithm. A potential candidate is the Reed-Solomon code. Should improve overall success level of communication sessions. Important aspect: this coding algorithm is suitable for sending of fixed-length data. Use of Reed-Solomon code would require a change in a design of the communication session: a data would be sent in chunks (basically a sequence of encoded messages).
- Knowledge related to Img. 3 Fletcher–Munson curves: how frequency-pitch relation changes perception of the loudness by human [2] can help in finding optimal frequency regarding acceptance by human ear.
- Whole channel is used ineffectively since only frequency of sine wave is used for information transmission, but amplitude is not. Possible solution – approach which would utilize both amplitude and frequency.
    - Author thought about applying some sort of QAM approach, but after further thinking he rejected this idea. QAM utilizes phase modulation, which eventually can’t be applied in sound domain. The phase is too precise measure. Microphones might not be sensitive enough to catch it, and loudspeakers might not be precise enough to produce proper phases.

- Whole channel is used ineffectively – continuation. It is worth trying to compose multiple harmonic sines into one sound signal. On the receiving side – use Fourier transform method to decompose a received sound back to harmonic sines. Next, detect frequencies of those decomposed sines. It would give increase in bandwidth directly proportional to number of composed sines. Generally, Fourier transform method would provide benefits even with single-sine sound, because it would enable the decoding algorithm to bypass irrelevant surrounding sounds and find only those which match operational spectrum. Implementation of solution based on Fourier transform method would require much more time and efforts.
- Try to implement Band-pass filter on the receiver side (programmatically). This filter would cut frequencies which exceed the operational spectrum. Use of Fourier transform method would make Band-pass filter method useless.
- On-streets experiment. Walk in public places for longer time and record surrounding sound. Analyze recorder frequencies spectrum. Find frequencies that occur the least in the surrounding. While this experiment wouldn't be useful for DataViaSound v1.0 implementation, it could be useful for implementation which involves frequencies decomposition.
- Perform tests which would check impact of battery level on success level of communication sessions. See subsubsection 2.2.5.15 for more detailed description of the problem. More complicated for test operator – requires constant involvement. Approach: run multiple sequences of tests, in the same way as control tests, but each test should be done on specific battery level. Check resulting charts of success level.
- Another potentially useful analysis approach: go through single test reports, detect those characters which are the most problematic for decoding. This process can be automated. This information could be used to tell which frequencies within operational binding are less suitable. As result, those frequencies should be excluded from binding.

Summary

This work was created as a summary of author's 3.5 year studies on Engineer (Bachelor) degree on Częstochowa University of Technology. As a basic aim of this work, author had decided to use and test a way of data transportation between electronic devices, which would be based only on use of sound. Another no less important aim was to achieve the best precision of data reception by a receiver.

Raised issue was a challenge for author primarily because it required analysis of wide range of parameters which could have an impact on quality of data transportation. For purpose of practical demonstration of newly created technology, author had decided to create Android application called DataViaSound. For statistical and study purposes of the given technology, author had created a dedicated tool for automated testing called DataViaSoundTester. This testing tool took a key role in finding of an optimal configuration of the application. Since operation of the technology was based on encoding of the data into a predefined set of frequencies, at the end of the work author had found a set of frequencies and loudness level which gave the highest statistical precision of data transportation for both devices. Besides, author had found and fixed bugs in the initial version of the application, which had very strong impact on precision of communication.

Text of the work contains an introduction to issues related to the discussed problem, presents analysis of the given problem, sequence of author's actions while attempting to solve it and conclusions inferred out of received results.

Since the depicted problem is complicated, there is still a perspective of future development of this technology and approach. Author assumes that work on technology of data via sound transportation can be continued in Master diploma work.

Streszczenie

Niniejsza praca powstała jako podsumowanie 3.5-letnich studiów inżynierskich autora na Politechnice Częstochowskiej. Jako zasadniczy cel tej pracy, autor postanowił wykorzystać i przetestować sposób przekazywania danych między urządzeniami elektronicznymi, który by opierał się wyłącznie na użyciu dźwięku. Nie mniej ważnym zadaniem było również osiągnięcie jak najlepszej precyzji odczytu danych przez odbiorcę.

Przedstawione zagadnienie stanowiło wyzwanie dla autora przede wszystkim z powodu konieczności analizy szerokiego zakresu parametrów mogących mieć wpływ na jakość przekazywania danych. Do celów naocznej demonstracji działania nowo-powstałej technologii, autor postanowił stworzyć aplikację Android pod nazwą DataViaSound. Do celów statystycznych i badawczych owej technologii, autor stworzył dedykowanie narzędzie do automatycznego testowania pod nazwą DataViaSoundTester. Narzędzie do testowania miało kluczową rolę w znalezieniu optymalnej konfiguracji aplikacji. Jako że działanie technologii opierało się na kodowaniu danych w predefiniowanym zestawie częstotliwości, pod koniec pracy autor znalazł taki zestaw częstotliwości i poziom głośności, które sprawiały największą statystyczną precyzję przekazywania danych dla obu testowanych urządzeń. Również warto dodać, że autor znalazł i naprawił błędy początkowej wersji aplikacji, które miały bardzo istotny wpływ na precyzję komunikacji.

Tekst pracy zawiera wprowadzenie do kwestii związanych z omawianym problemem, przedstawia analizę rozważanego problemu, tok postępowania autora przy podjętych próbach jego rozwiązania oraz wnioski z otrzymanych wyników.

Jako że przedstawiony problem jest skomplikowany, wciąż zostaje perspektywa przyszłego rozwoju tej technologii i opracowanego podejścia. Autor zakłada, że praca nad technologią przekazywania danych za pomocą dźwięku może być kontynuowana w pracy magisterskiej.

References

[1] *What is an Android Activity?*, relevant for December 10, 2018, source: http://programmerguru.com/android-tutorial/android-activity/

[2] *Audio engineering: minimal audible sound levels*, relevant for December 10, 2018, source: https://www.webel.com.au/content/audio-engineering-minimal-audible-sound-levels

[3] *Gingerbread feature: Near Field Communication*, relevant for January 06, 2019, source: https://www.androidcentral.com/gingerbread-feature-near-field-communication

[4] *JAVA TECHNOLOGY: THE EARLY YEARS*, web archive, source: https://web.archive.org/web/20050420081440/http://java.sun.com/features/1998/05/birthday.html

[5] *Java Comunity Process home page*, relevant for January 06, 2019, source: https://www.jcp.org/en/home/index

[6] *Announcing the Android 1.0 SDK, release 1*, relevant for January 06, 2019, source: https://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html

[7] *Xperia Support Forum: Z5 mic hole???*, relevant for January 06, 2019, source: https://talk.sonymobile.com/t5/Xperia-Z5-Z5-Compact-Z5-Premium/Z5-mic-hole/td-p/1121964, archived version of official overview on Sony web page: https://web.archive.org/web/20160406115950/http://support.sonymobile.com/global-en/xperiaz5/userguide/Overview-SE/

[8] *Sony Xperia Z5 review*, relevant for January 06, 2019, source: https://www.techradar.com/reviews/phones/mobile-phones/sony-xperia-z5-1303152/review

[9] *Geekbench 4 CPU Search Xperia Z5*, relevant for January 06, 2019, source: https://browser.geekbench.com/v4/cpu/search?q=Xperia+Z5

[10] *Lenovo A1000 full phone specifications*, relevant for January 06, 2019, source: https://www.gsmarena.com/lenovo_a1000-7605.php

[11] *Geekbench 4 CPU Search Lenovo a1000*, relevant for January 06, 2019, source: https://browser.geekbench.com/v4/cpu/search?q=Lenovo+a1000

Appendix A. Code snippet from DataViaSound v1.0

```
package com.luckynick.android.test;
import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioTrack;
import android.media.MediaPlayer;
import android.os.Build;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import static com.luckynick.custom.Utils.*;
public class SoundGenerator {
    public interface Listener {
        void playStopped();
    }
    private String recordPath;
    public static final String LOG_TAG = "Generator";
    public static List<Listener> playStoppedSubs = new ArrayList<>();
    SoundGenerator(String rec)
    {
        recordPath = rec;
    }
    public void playMessage(int freqBindingBase,
                            double freqBindingScale, String m)
    {
        int frequenciesArr[] = BaseActivity.getFreqBinding(freqBindingScale);
        String message = JUNK_RIGHT + START_TAG;
        message += toHex(m);
        message += END_TAG + JUNK_RIGHT;
        Log(LOG_TAG, "Playing new message: " + message);
        if(m.equals("")) return;
        int numSamples = SAMPLE_RATE * BEEP_DURATION / 1000;
        double[][] mSound = new double[message.length()][numSamples];
        short[][] mBuffer = new short[message.length()][numSamples];
        for (int i = 0; i < message.length(); i++) {
            int index = (int)message.charAt(i);
            double currentFreq;
            if(index >= frequenciesArr.length)
                currentFreq = frequenciesArr[ERROR_CHAR] + freqBindingBase;
            else currentFreq = frequenciesArr[message.charAt(i)]
```

```
                    + freqBindingBase;
            if(currentFreq == -1.0) currentFreq =
                    frequenciesArr[ERROR_CHAR] + freqBindingBase;
            for (int j = 0; j < mSound[i].length; j++) {
                mSound[i][j] = Math.sin(2.0 * Math.PI *
                        j / (SAMPLE_RATE / currentFreq));
                mBuffer[i][j] = (short) (mSound[i][j]
                        * Short.MAX_VALUE);
            }
        }
        AudioTrack mAudioTrack = new AudioTrack(AudioManager.STREAM_MUSIC,
                SAMPLE_RATE, AudioFormat.CHANNEL_OUT_MONO,
                AudioFormat.ENCODING_PCM_16BIT, BUFFER_SIZE_OUT,
                AudioTrack.MODE_STREAM);
        mAudioTrack.setStereoVolume(AudioTrack.getMaxVolume(),
                AudioTrack.getMaxVolume());
        for(short[] arr : mBuffer)
        {
            play(mAudioTrack, arr);
        }
        mAudioTrack.release();
        for (Listener sub : playStoppedSubs) {
            sub.playStopped();
        }
    }
    public void playMessage(int freqBindingBase, double freqBindingScale,
                            String m, final int loudnessLevel,
                            boolean wrapInTags)
    {
        int frequenciesArr[] = BaseActivity.getFreqBinding(freqBindingScale);
        String message = JUNK_RIGHT + START_TAG; //junk here for test
        message += toHex(m);
        message += END_TAG + JUNK_RIGHT + JUNK_RIGHT;
        if(!wrapInTags) message = toHex(m);
        Log(LOG_TAG, "Playing new message: " + message);
        if(m.equals("")) return;
        int numSamples = SAMPLE_RATE * BEEP_DURATION / 1000;
        double[][] mSound = new double[message.length()][numSamples];
        short[][] mBuffer = new short[message.length()][numSamples];
        for (int i = 0; i < message.length(); i++) {
            int index = (int)message.charAt(i);
            double currentFreq;
            if(index >= frequenciesArr.length) currentFreq =
                    frequenciesArr[ERROR_CHAR] + freqBindingBase;
```

```
            else currentFreq = frequenciesArr[message.charAt(i)]
                    + freqBindingBase;
            if(currentFreq == -1.0) currentFreq = frequenciesArr[ERROR_CHAR]
                    + freqBindingBase;
            for (int j = 0; j < mSound[i].length; j++) {
                mSound[i][j] = Math.sin(2.0 * Math.PI *
                        j / (SAMPLE_RATE / currentFreq));
                mBuffer[i][j] = (short) (mSound[i][j] * Short.MAX_VALUE);
            }
        }
        AudioTrack mAudioTrack = new AudioTrack(
                AudioManager.STREAM_MUSIC, SAMPLE_RATE,
                AudioFormat.CHANNEL_OUT_MONO,
                AudioFormat.ENCODING_PCM_16BIT,
                BUFFER_SIZE_OUT, AudioTrack.MODE_STREAM);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            float log1=(float)(Math.log(100-loudnessLevel)/Math.log(100));
            mAudioTrack.setVolume(1-log1);
        }
        else {
            float deviceLoudness = (loudnessLevel / 100.0f) *
                    AudioTrack.getMaxVolume();
            mAudioTrack.setStereoVolume(deviceLoudness, deviceLoudness);
        }
        for(short[] arr : mBuffer)
        {
            play(mAudioTrack, arr);
        }
        mAudioTrack.release();
        for (Listener sub : playStoppedSubs) {
            sub.playStopped();
        }
    }
    public void play(short[] buffer)
    {
        AudioTrack mAudioTrack = new AudioTrack(
                AudioManager.STREAM_MUSIC, SAMPLE_RATE,
                AudioFormat.CHANNEL_OUT_MONO,
                AudioFormat.ENCODING_PCM_16BIT,
                BUFFER_SIZE_OUT, AudioTrack.MODE_STREAM);
        mAudioTrack.setStereoVolume(AudioTrack.getMaxVolume(),
                AudioTrack.getMaxVolume());
        mAudioTrack.play();
        mAudioTrack.write(buffer, 0, buffer.length);
```

```
            mAudioTrack.stop();
            mAudioTrack.release();
        }
        public void play(AudioTrack mAudioTrack, short[] buffer)
        {
            if(buffer == null) System.out.println("buffer is null");
            mAudioTrack.play();
            mAudioTrack.write(buffer, 0, buffer.length);
            mAudioTrack.stop();
        }
        public boolean playMediaPlayer()
        {
            if(!new File(recordPath).exists()) return false;
            MediaPlayer mp = new MediaPlayer();
            mp.setAudioStreamType(AudioManager.STREAM_MUSIC);
            try {
                mp.setDataSource(recordPath);
                mp.prepare();
            } catch (IOException e) {
                e.printStackTrace();
            }
            mp.start();
            while(mp.isPlaying())
            {
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            mp.release();
            return true;
        }
        public static void subscribePlayStoppedEvent(Listener sub) {
            playStoppedSubs.add(sub);
        }
}
```

Appendix B. Content of attached CD disk

Diploma.pdf – this document

DataViaSound – folder with source code of the Android application

DataViaSoundTester – folder with source code of the testing program

List of illustrations

Index

Imię i nazwisko: ……... Częstochowa, dn. ……..

Nr albumu: …….

Kierunek: ……

Wydział: …….

Politechnika Częstochowa

OŚWIADCZENIE

autora pracy dyplomowej*

Pod rygorem odpowiedzialności karnej oświadczam, że złożona przeze mnie praca dyplomowa pt. …….....

……
….

jest moim samodzielnym opracowaniem i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Jednocześnie oświadczam, że praca w całości lub we fragmentach nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w uczelni.

Wyrażam/ nie wyrażam** zgodę na nieodpłatne wykorzystanie przez Politechnikę Częstochowską całości lub fragmentów ww. pracy w publikacjach Politechniki Częstochowskiej.

Ponadto oświadczam, że treść pracy przedstawionej przeze mnie do obrony zawarta na przekazywanym nośniku elektronicznym jest identyczna z wersją drukowaną.

……
podpis studenta

* w przypadku zbiorowej pracy dyplomowej, dołącza się oświadczenia każdego ze współautorów pracy dyplomowej

** niepotrzebne skreślić