1. Kodowanie danych

Dane są kodowane jako częstotliwości dźwięku. Ze względu na małe dostępne pasmo konieczne były przesyłanie połówek bajtów. Kolejne bity danych są odwzorowane w następujący sposób:

Bit 0 – 19/20kHz, bit 1 – 21kHz, bit 2 – 22kHz, bit 3 – 23kHz. W celu uniknięcia problemu z odróżnieniem liczby 0 od ciszy, bit 0 został podzielony na 2 częstotliwości: 0 to 19kHz, a 1 to 20kHz.

Kolejne bity są ze sobą sumowane i w ten sposób np. liczba 9 (1001) jest przestawiona jako fala dźwiękowa ze składowymi częstotliwościami: 20kHz, 23kHz.

2. Generowanie próbek

Próbki przetrzymywane są jako tablice bajtów (PC) lub shortów (Android) o długości równej długości wektora FFT. Główne próbki (pojedyncze bity) są wyliczane. Każda pojedyncza próbka w tablicy jest wyznaczana jako:

$$sample[i] = \sin\frac{2\pi i * freq}{sampleRate} * maxValue$$

gdzie:

i – indeks kolejnej próbki
freq – częstotliwość próbki
sampleRate – częstotliwość próbkowania dźwięku
maxValue – maksymalna wartość danego typu (byte – 127, short – 32767)

Pozostałe próbki są wyliczane jako sumy głównych próbek tak, jak zostało to opisane powyżej.

3. Wysyłanie i odbieranie dźwięku

Klasy odpowiedzialne: SenderPC, ReceiverPC oraz Sender, Receiver (Android)

Dźwięk jest wysyłany i odbierany przez SourceDataLine i TargetDataLine (PC) oraz AudioRecord i AudioTrack (Android).

Wysyłanie odbywa się przez wpisanie całej próbki do odpowiedniego strumienia.

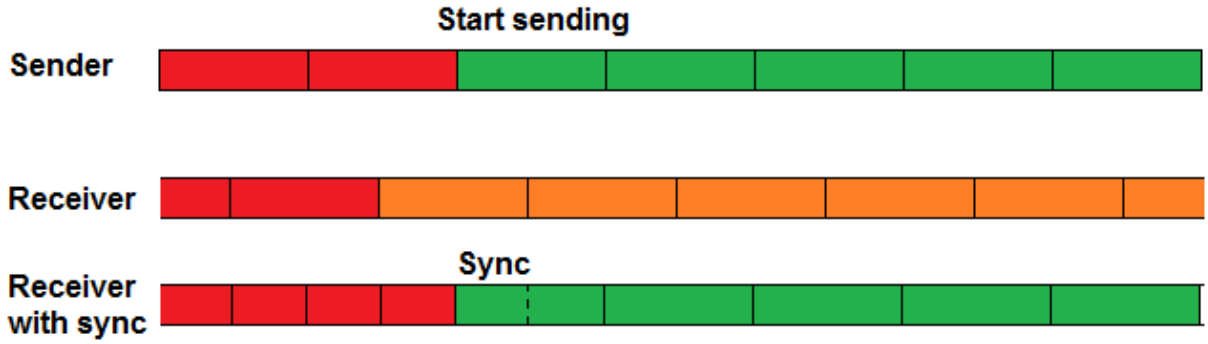
Przy odbieraniu próbki są wczytywane do bufora, a następnie przetwarzane.

4. Przetwarzanie próbek

Klasy odpowiedzialne: ReceiverPC, Receiver

W celu odczytania danych z kolejnych próbek konieczna jest wykorzystanie transformaty Fouriera. Dzięki niej z kolejnych próbek otrzymujemy spektrum częstotliwości (co kilkadziesiąt herców).

Dodatkowo, żeby transformata dostawała na wejściu (z jak najmniejszym błędem) ten sam bufor co był wysłany, przed rozpoczęciem właściwego odbierania wykonywana synchronizacja przez obliczanie transformaty o mniejszej długości wektora.



Po zsynchronizowaniu, klasa Receiver zaczyna działać w normalnym trybie. Dla każdego kolejnego bufora po wyznaczeniu spektrum sprawdza czy występuje dana częstotliwość i zgodnie z kodowaniem odczytuje kolejne bity.

5. Ramka

Klasy odpowiedzialne: AbstractDataReader, PCDataReader, AndroidDataReader oraz DataProcessor

Po odczytaniu danych przez Receiver są one przekazywane do DataReader'a. Jego zadaniem jest wywolanie DataProcessora, obliczenie CRC i wywołanie callbacków onSuccess lub onFailure.

DataProcessor jest natomiast maszyna stanową. Jej zadaniem jest wyszukanie ramki oraz reagowanie na błedy.

W celu uniknięcia problemów z echem każda próbka jest wysyłana 5 razy.

Ramka składa się z nagłówka (służy do synchronizacji początku i końca danych), danych oraz 16-bitowej sumy kontrolnej CRC.

Nagłówek składa się z próbek: cisza x5, 0 (19kHz) x5, cisza x5. Tolerancja błędów w tym przypadku jest bardzo mała. Maksymalna tolerancją jest możliwość wystąpienia tych składowych 4 (a nie 5) razy, ale nie mogą wystąpić w tym czasie żadne inne częstotliwości.

Dla danych tolerancja jest znacznie większa. Brana jest ta liczba, która pojawi się przynajmniej 3 z 5 razy.

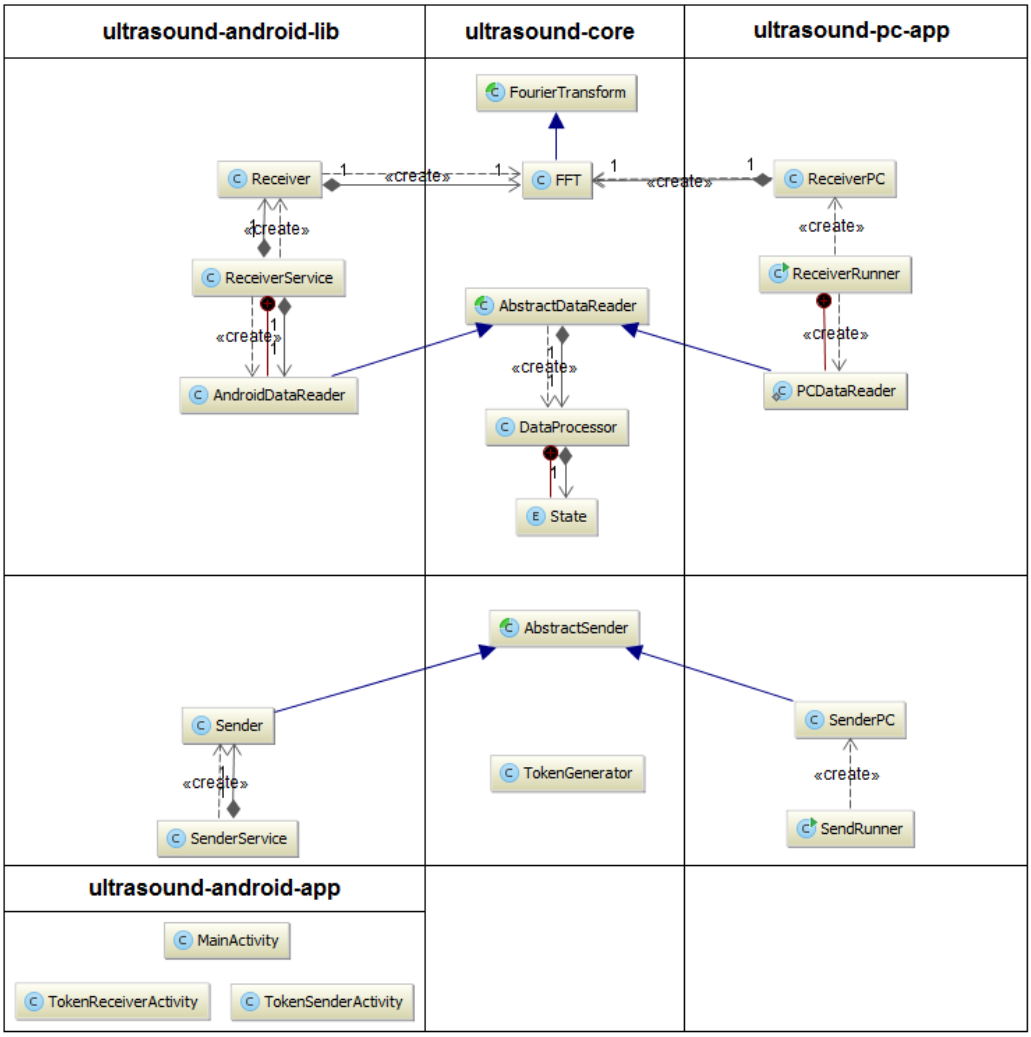
Na koniec obliczana jest suma CRC i porównywana z przetworzonymi danymi.

6. Ograniczenia

Największy ograniczeniem w projekcie jest maksymalna częstotliwość próbkowania dźwięku na telefonach z Androidem. Próbkowanie 48kHz pozwoliło wykorzystać pasmo 5kHz. Dodatkowo operowanie na częstotliwościach bliskich maksymalnej, bardzo zmniejsza dokładność obliczeń.

7. Dalszy rozwój
    a. Zwiększenie tolerancji błędów, zapamiętanie różnych kombinacji danych i sprawdzenie czy któraś z nich jest poprawna przez porównanie z CRC.
    b. Dodanie do nagłówka dodatkowych danych takich jak długość ramki, czy ID odbiorcy.
    c. Zwiększenie zasięgu działania.

Diagram klas



8. Struktura serwisów

ReceiverService

- Uruchamianie i zatrzymywanie serwisu poprzez intent

```
private void startReceiverService() {
    Intent intent = new Intent(this, ReceiverService.class);
    startService(intent);
}

private void stopReceiverService() {
    Intent intent = new Intent(this, ReceiverService.class);
    stopService(intent);
}
```

- Aby otrzymywać wyniki należy zarejestrować odpowiedni BroadcastReceiver

```
private BroadcastReceiver serviceBroadcastReceiver = (context, intent) -> {
        Bundle bundle = intent.getExtras();
        if (bundle != null) {
            byte[] tokenByteArray = bundle.getByteArray(ReceiverService.BYTE_BUFFER_KEY);
            updateToken(tokenByteArray);
        }
};
```

SenderService

- Uruchamianie przy użyciu Intentu, powinien dodatkowo zostać załączony Token

```
private void startSenderService(String tokenString) {
    byte[] tokenData = TokenGenerator.convertFromString(tokenString);

    Intent intent = new Intent(this, SenderService.class);
    intent.putExtra(SenderService.BYTE_BUFFER_KEY, tokenData);
    startService(intent);
}
```

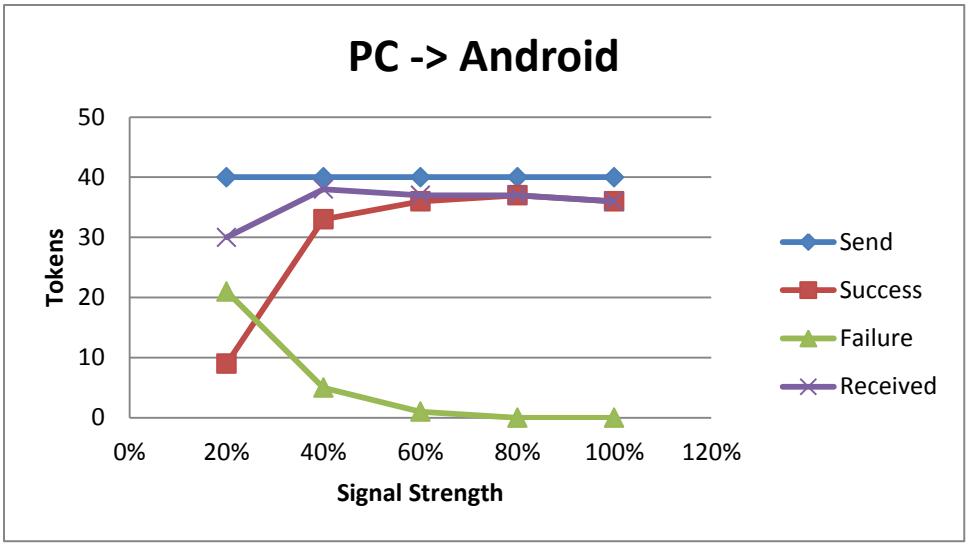
- Zatrzymywanie przy użyciu Intentu

```
private void stopSenderService() {
    Intent intent = new Intent(this, SenderService.class);
    stopService(intent);
}
```

9. Testy

Przeprowadzona kilka testów dla różnej siły sygnału emitującego ultradźwięki dla dwóch przypadków. Komunikacja z Androida na PC oraz z PC na Androida.

PC -> Android				
Signal	Send	Success	Failure	Received
100%	40	36	0	36
80%	40	37	0	37
60%	40	36	1	37
40%	40	33	5	38
20%	40	9	21	30



Android -> PC				
Signal	Send	Success	Failure	Received
100%	40	2	27	29
80%	40	0	18	18
60%	40	0	17	17
40%	40	0	13	13
20%	40	0	10	10