2020

Transmisja w systemie FEC

MACIEJ BRONIKOWSKI 248838
SZYMON PLEŚNIEROWICZ 248887

1 CEL I ZAŁOŻENIA PROJEKTU

Projekt zakładał porównanie kodów nadmiarowych FEC (Forward Error Correction). Całość projektu została zrealizowana w języku c++, na systemie operacyjnym Linux Ubuntu, z pomocą biblioteki ezpwd. Biblioteka korzysta z wbudowanego w system operacyjny narzędzia generującego kod nadmiarowy BCH, oraz wprowadza swoją własną implementację kodu RS.

System, w którym przeprowadzano doświadczenia można przedstawić na rysunku:

1.1 DANE NADAWANE

Dane nadawane generowane są przez system. Wielkość segmentu danych jest ustalona jako największa możliwa ilość bitów dla danych parametrów kodu BCH. Dla przejrzystości wyników kod RS został przygotowany w taki sposób, aby ilość bitów nadmiarowych zgadzała się z tą w kodzie BCH. Ilość wygenerowanych danych jest wprowadzana podczas wykonywania programu.

1.2 KODER/DEKODER

Kodowanie i dekodowanie w systemie zostało zrealizowane za pomocą 3 kodów nadmiarowych:

- BCH
- RS
- Potrojeniowy

Kod potrojeniowy jest prostym kodem korygowania błędów. Polega na powieleniu każdego bitu danych nadawanych tak, aby każdemu bitowi przed zakodowaniem odpowiadały 3 bity po zakodowaniu. Korekcja polega na wybraniu na podstawie 3 bitów danych zakodowanych, takiej wartości bitu kodu, którego jest więcej. Przykładowo dla danych 101 zostanie wybrany bit 1. Z założenia jest to kod wymagający naprawdę dużej ilości bitów nadmiarowych. Dla n bitów danych zostanie wygenerowane 3n bitów zakodowanych.

Kod BCH wymaga znacznie mniej danych nadmiarowych (w zależności od doboru stopnia wielomianu generującego kod). Jednak implementacja kodu wymaga znacznie większej wiedzy. Należy również pamiętać o dobieraniu odpowiednich parametrów kodu BCH. Kod BCH można przedstawić za pomocą 3 parametrów: n-długość wektora kodowego, k-długość komunikatu, oraz t-zdolności korekcji błędów.

Kod RS jest podklasą kodów BCH niebinarnych nad ciałem GF(q). Są one znacznie lepsze w korygowaniu błędów grupowych. Kod ten może zostać przedstawiony za pomocą dwóch parametrów: n-długość wektora kodowego, oraz k-długość komunikatu.

1.3 KANAŁ TRANSMISYJNY

Przesyłanie kanałem transmisyjnym zostało zasymulowane na podstawie dwóch modeli kanałów transmisyjnych:

-model BSC
- model Gilberta

Model BSC zakłada, że podczas przesyłania danych, za możliwość wystąpienia błędu odpowiada tylko jeden parametr – prawdopodobieństwo wystąpienia tego błędu. Przykładowo dla transmisji głosu w systemach GSM wymagane jest uwzględnienie błędów o prawdopodobieństwie P= 10^{-3}.

Model Gilberta zakłada istnienie dwóch stanów: stanu, w którym odebrane dane są identyczne do tych nadanych, oraz stanu, w którym dane są wartościami błędnymi. Za przejścia między tymi danymi odpowiadają dwa parametry – prawdopodobieństwo przejścia modelu w stan zły i prawdopodobieństwo przejścia systemu w stan dobry. Model ten wykorzystywany jest przykładowo podczas projektowania systemu łączności satelitarnej, oraz łączności internetowej na duże odległości.

1.4 DANE ODEBRANE

Dane odebrane są porównywane do tych sprzed wysłania, a następnie jest wyliczana elementowa stopa błędów BER. Wynikiem są również nadmiarowości poszczególnych błędów.

2 OPIS WYKONANEGO NARZĘDZIA

2.1 OPIS NARZĘDZI UDOSTĘPNIANYCH PRZEZ BIBLIOTEKĘ

Projekt został wykonany na podstawie biblioteki ezpwd, która wykorzystuje wbudowany system korekcji błędów linuxa. Wykorzystane zostały dwa elementy tej biblioteki: kodowanie i dekodowanie w kodzie BCH, oraz kodowanie i dekodowanie w kodzie RS.

2.1.1 BCH

Pierwszym etapem jest stworzenie obiektu klasy BCH z odpowiednimi parametrami:
ezpwd::BCH<n,k,t> bch_codec;
gdzie odpowiednio n oznacza długość wektora kodowego, k oznacza długość komunikatu, oraz t oznacza zdolność korekcyjną kodu.

Następnie udostępniane są dwie najważniejsze metody odnośnie tej klasy:
bch_codec.encode(codeword) – kodowanie sygnału
bch_codec.decode(codeword) – dekodowanie sygnału wraz z korekcją błędów.

2.1.2 RS

Tworzenie obiektu kodera RS wygląda w podobny sposób. Należy najpierw stworzyć klasę kodera:
ezpwd::RS<n,k> rs;
gdzie n oznacza ilość symboli, a k oznacza długość komunikatu

Następnie udostępniane są dwie najważniejsze metody tej klasy:
rs.encode(data) - kodowanie sygnału
rs.decode(data) – dekodowanie sygnału oraz korekcja błędu.

2.2 IMPLEMENTACJA KODU POTROJENIOWEGO

Koder oraz dekoder kodu potrojeniowego został zaimplementowany przez nas. Odpowiada za nią klasa tripling w plikach tripling.h oraz tripling.cpp. Implementacja opiera się na prostym schemacie. Koder przesuwa każdy bit danych do zakodowania o 2 pozycje, a następnie uzupełnia miejsce kopią wartości bitu przesuniętego. Dekodowanie opiera się na pobraniu trzech kolejnych bitów, wybraniu odpowiedniej wartości na podstawie tego, której wartości było więcej w kodzie i uzupełnianie kolejnych pozycji danych zdekodowanych tą wartością. Klasa udostępnia dwie ważne metody:
encode(data) – zakodowanie wartości
decode(data) – zdekodowanie wartości

2.3 IMPLEMENTACJA MODELÓW KANAŁÓW TRANSMISYJNYCH

Kanały transmisyjne również zostały zaimplementowane przez nas. W projekcie zostały zawarte dwa modele kanałów transmisyjnych: model BSC, oraz model Gilberta.

2.3.1 Model BSC

Przesyłanie danych kanałem BSC symuluje klasa Bsc, zawarta w plikach Bsc.h, oraz Bsc.cpp. Udostępnia ona metodę
noise(data, prop)

Na podstawie prop – prawdopodobieństwa zmiany stanu bitu w promilach, dla kolejnych bitów data sprawdzana jest wartość losowania i jeżeli wylosowano wartość mniejszą niż prawdopodobieństwo, stan bitu jest negowany.

2.3.2 Model Gilberta

Kanał Gilberta symulowany jest dzięki klasie Gilbert, zawartej w plikach Gilbert.h, oraz Gilbert.cpp. Klasa udostępnia metodę
noideG(data, prop1, prop2)

Na podstawie prop1 – prawdopodobieństwo przejścia systemu w stan zły i prop 2 – prawdopodobieństwo tego, że system pozostanie w złym stanie, dla kolejnych bitów losowana jest informacja, czy system zmienia stan. Jeżeli stan jest tym złym, kolejne bity są negowane, jeżeli dobrym, wartość bitów pozostaje niezmienna.

2.4 PARAMETRY MODELI KANAŁÓW I KODERÓW

2.4.1 Parametry koderów

Z racji implementacji biblioteki, wartości parametrów koderów muszą być definiowane przed kompilacją programu, dlatego możliwość ich doboru zawarta jest w pliku menu.h, który wraz z plikem menu.cpp odpowiada za złożenie działania programu w całość. Możliwe jest ustawienie 3 parametrów

```
CODE_N – długość słowa kodowego
CODE_K – długość bloku danych do zakodowania
CODE_T – zdolność korekcyjna kodu BCH
```

2.4.2 Parametry modeli kanałów

Możliwość ustawienia parametrów kanałów transmisyjnych możliwa jest do dobrania z pozycji uruchomionego programu.

2.5 Dane nadawane

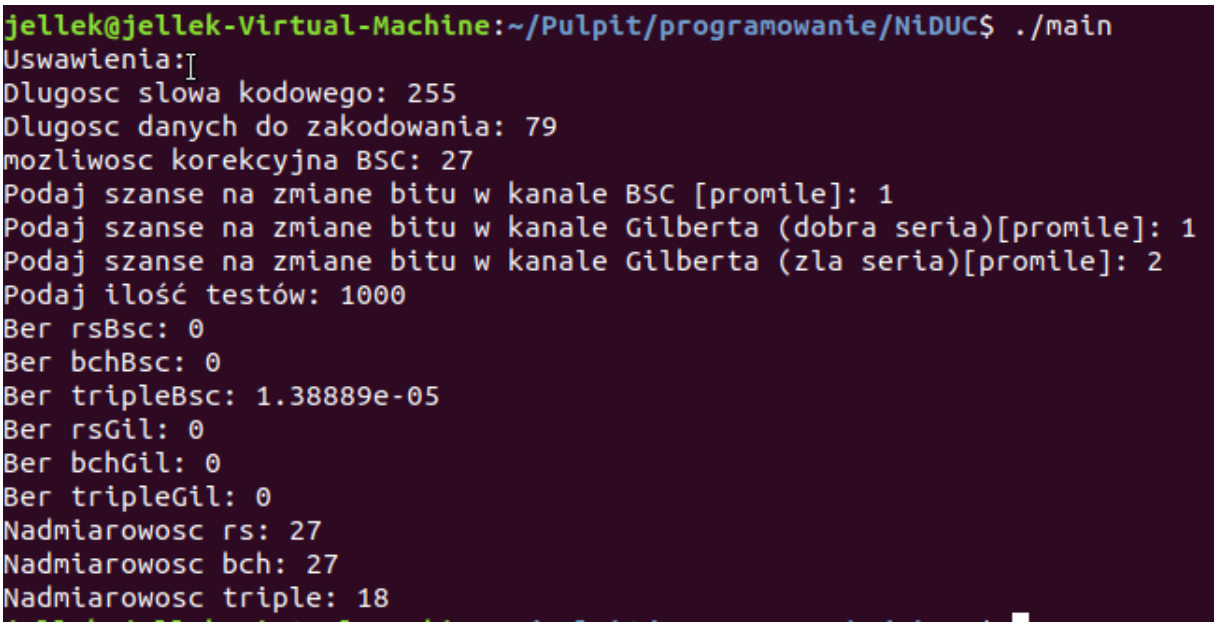
Dane nadawane są przechowywane w strukturze klasy vector, której elementami są 8 bitowe elementy. Odpowiada to symulacji wysyłania znaków w systemie. Dla każdego kolejnego testu danymi są inne wartości, losowane za pomocą funkcji rand().

2.6 Kompilacja i Uruchomienie programu

Aby skompilować program należy użyć polecenia make main. Po użyciu komendy, środowisko kompiluje pliki main.cpp, wszystkie pliki zawarte w folderze src, oraz nagłówki koderów RS i BCH z biblioteki ezpwd do pliku main.

Aby uruchomić program należy wydać polecenie ./main w środowisku linux.

2.6.1 Opisz uruchomionego programu



Podczas uruchomienia programu przedstawiane są najpierw parametry koderów ustawione w pliku main.h.
Kolejnym etapem programu jest wprowadzenie wartości parametrów modeli kanałów transmisyjnych w promilach kolejno dla kanału w modelu BSC, oraz kanału w modelu Gilberta. Dla modelu Gilberta należy podać dwa parametry, promil szansy przejścia do stanu złego i promil szansy pozostania w stanie złym kanału.
Następnym etapem jest podanie wartości ilości testów. Odpowiada ona za ilość kolejnych testów dla wszystkich modeli kanałów i wszystkich koderów.
Ostatnim etapem jest przedstawienie średniej wartości BER dla wszystkich kanałów i wszystkich koderów po zdekodowaniu sygnału, obliczonej z wszystkich wykonanych testów, oraz przedstawienie nadmiarowości w bajta wszystkich danych.

3 ORGANIZACJA EKSPERYMENTU

Eksperyment polegał na kolejnym nadawaniu różnych parametrów koderów symulatora, oraz ustawianiu kolejnych parametrów modeli kanałów transmisyjnych. Parametry koderów symulatora można przedstawić za pomocą tabeli:

	Eksp. 1-4	Eksp. 5-8	Eksp. 9-12	Eksp. 13-16	Eksp. 17-20	Eksp. 21-24
Parametr n	255	255	255	255	255	255
Parametr k	239	231	187	179	108	79
Parametr t	2	3	9	10	22	29

Dane modeli kanałów transmisyjnych można przedstawić za pomocą tabeli:

	Eksp. 1,5,9,13,17,21	Eksp. 2,6,10,14,18,22	Eksp. 3,7,11,15,19,23	Eksp. 4,8,12,16,20,24
Kanał BSC	1 ‰	3 ‰	5 ‰	15 ‰
Kanał Gilberta dobra seria	1 ‰	3 ‰	5 ‰	15 ‰
Kanał Gilberta zła seria	2 ‰	5 ‰	10 ‰	20 ‰

Długość danych była ustalana na podstawie maksymalnej długości segmentów danych dla koderów bch i rs.

Wartości danych przed zakodowaniem były losowane za pomocą funkcji rand(), a wynikiem symulatora były kolejne wartości BER dla podanych w tabelach danych eksperymentu, obliczane na podstawie zestawienia danych przed nadaniem, oraz danych po zdekodowaniu. Każdy eksperyment był przeprowadzany przez symulator 1000 razy, a wartości BER wynikowe były średnią wartością BER każdego eksperymentu.

Dodatkowo system wyliczał i wyświetlał nadmiarowość w bajtach dla każdego z koderów. Nadmiarowości koderów BCH, oraz RS były zawsze tej samej wartości dla przejrzystości wyników. Nadmiarowość kodu potrojeniowego była 2 razy większa, niż długość słowa nadawanego.

4 WYNIKI

4.1 TABELA WYNIKÓW KANAŁU MODELU BSC

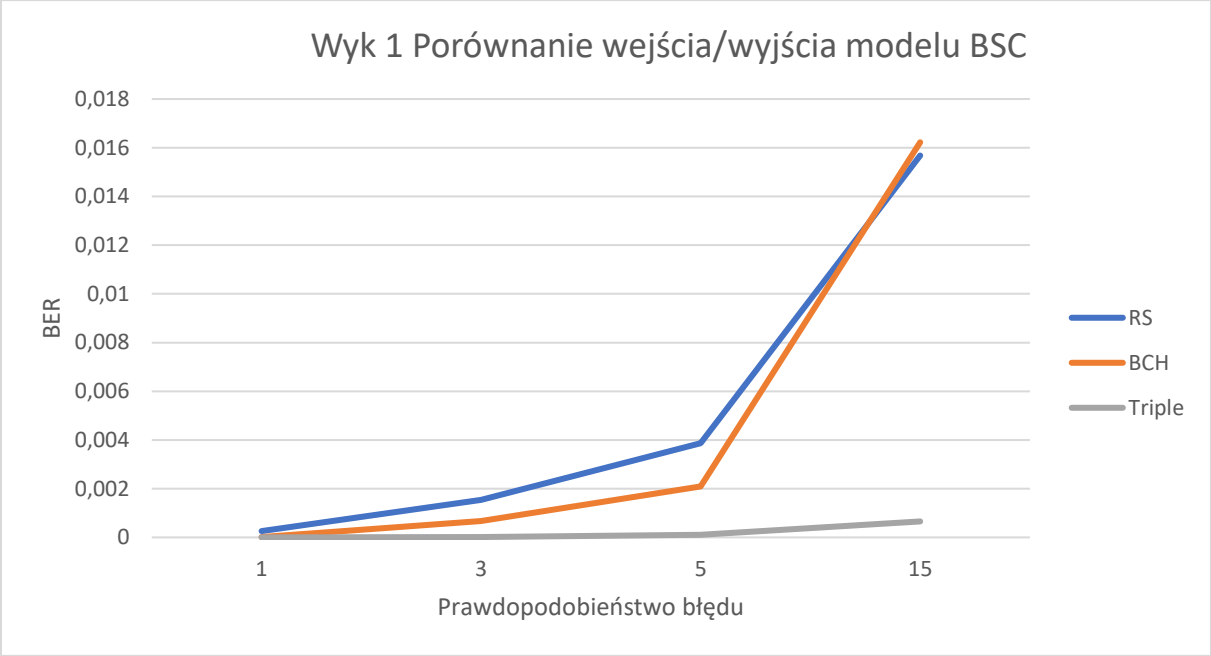
Eksperyment	1	2	3	4	Nadmiarowość
Ber RS:	0,000259	0,001534	0,003862	0,015681	2
Ber BCH:	0,000013	0,000668	0,002095	0,016224	2
Ber Triple:	0,000000	0,000013	0,000108	0,000655	58
Eksperyment	5	6	7	8	Nadmiarowość
Ber RS:	0,000188	0,001679	0,004138	0,015714	3
Ber BCH:	0,000018	0,000138	0,000540	0,011817	3
Ber Triple:	0,000004	0,000045	0,000054	0,000536	56
Eksperyment	9	10	11	12	Nadmiarowość
Ber RS:	0,000000	0,000011	0,000196	0,006647	9
Ber BCH:	0,000000	0,000000	0,000000	0,000179	9
Ber Triple:	0,000000	0,000033	0,000103	0,000707	46
Eksperyment	13	14	15	16	Nadmiarowość
Ber RS:	0,000000	0,000000	0,000080	0,003750	10
Ber BCH:	0,000000	0,000000	0,000000	0,000045	10
Ber Triple:	0,000006	0,000028	0,000108	0,000676	44
Eksperyment	17	18	19	20	Nadmiarowość
Ber RS:	0,000000	0,000000	0,000000	0,000000	22
Ber BCH:	0,000000	0,000000	0,000000	0,000000	22
Ber Triple:	0,000000	0,000038	0,000115	0,000856	26
Eksperyment	21	22	23	24	Nadmiarowość
Ber RS:	0,000000	0,000000	0,000000	0,000000	27
Ber BCH:	0,000000	0,000000	0,000000	0,000000	27
Ber Triple:	0,000000	0,000000	0,000111	0,000625	18

4.2 Tabela wyników kanału modelu Gilberta

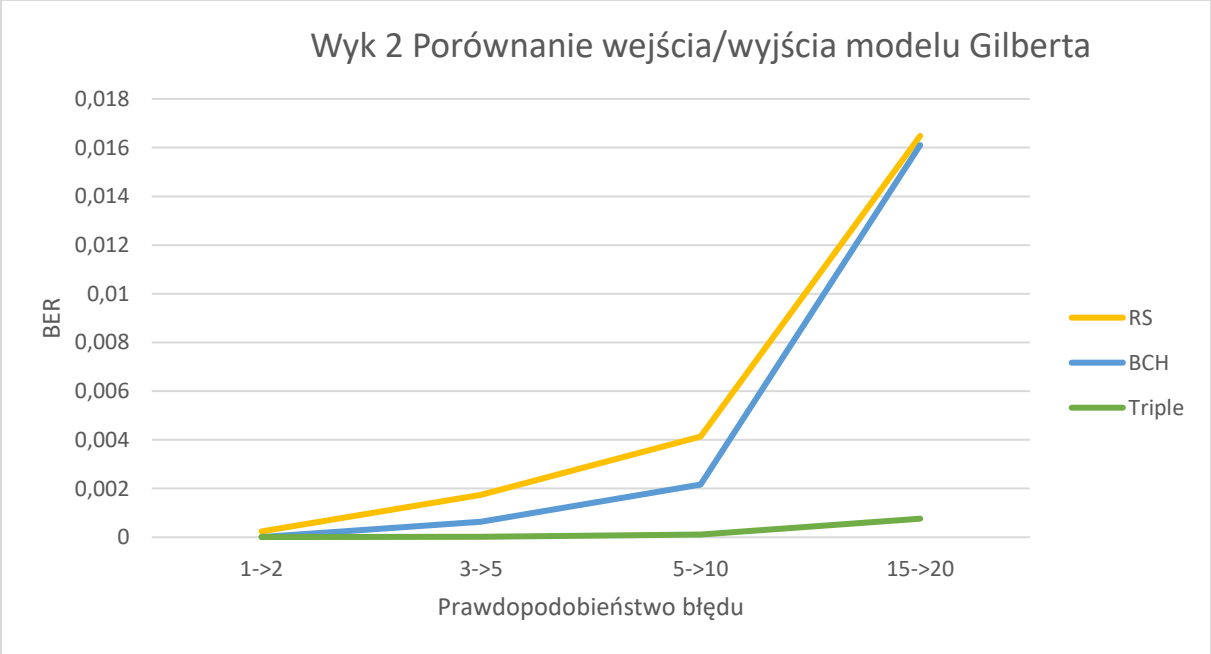
Eksperyment	1	2	3	4	Nadmiarowość
Ber RS:	0,000241	0,001746	0,004129	0,016483	2
Ber BCH:	0,000009	0,000638	0,002168	0,016103	2
Ber Triple:	0,000009	0,000022	0,000116	0,000763	58
Eksperyment	5	6	7	8	Nadmiarowość
Ber RS:	0,000241	0,001982	0,004174	0,016580	3
Ber BCH:	0,000000	0,000129	0,000777	0,011335	3
Ber Triple:	0,000000	0,000031	0,000112	0,000804	56
Eksperyment	9	10	11	12	Nadmiarowość
Ber RS:	0,000000	0,000000	0,000163	0,007022	9
Ber BCH:	0,000000	0,000000	0,000000	0,000114	9
Ber Triple:	0,000000	0,000038	0,000163	0,000755	46
Eksperyment	13	14	15	16	Nadmiarowość
Ber RS:	0,000000	0,000000	0,000045	0,004097	10
Ber BCH:	0,000000	0,000000	0,000000	0,000091	10
Ber Triple:	0,000006	0,000023	0,000148	0,000830	44
Eksperyment	17	18	19	20	Nadmiarowość
Ber RS:	0,000000	0,000000	0,000000	0,000058	22
Ber BCH:	0,000000	0,000000	0,000000	0,000000	22
Ber Triple:	0,000000	0,000048	0,000087	0,000731	26
Eksperyment	21	22	23	24	Nadmiarowość
Ber RS:	0,000000	0,000000	0,000000	0,000000	27
Ber BCH:	0,000000	0,000000	0,000000	0,000000	27
Ber Triple:	0,000000	0,000000	0,000069	0,000667	18

5 Analiza wyników

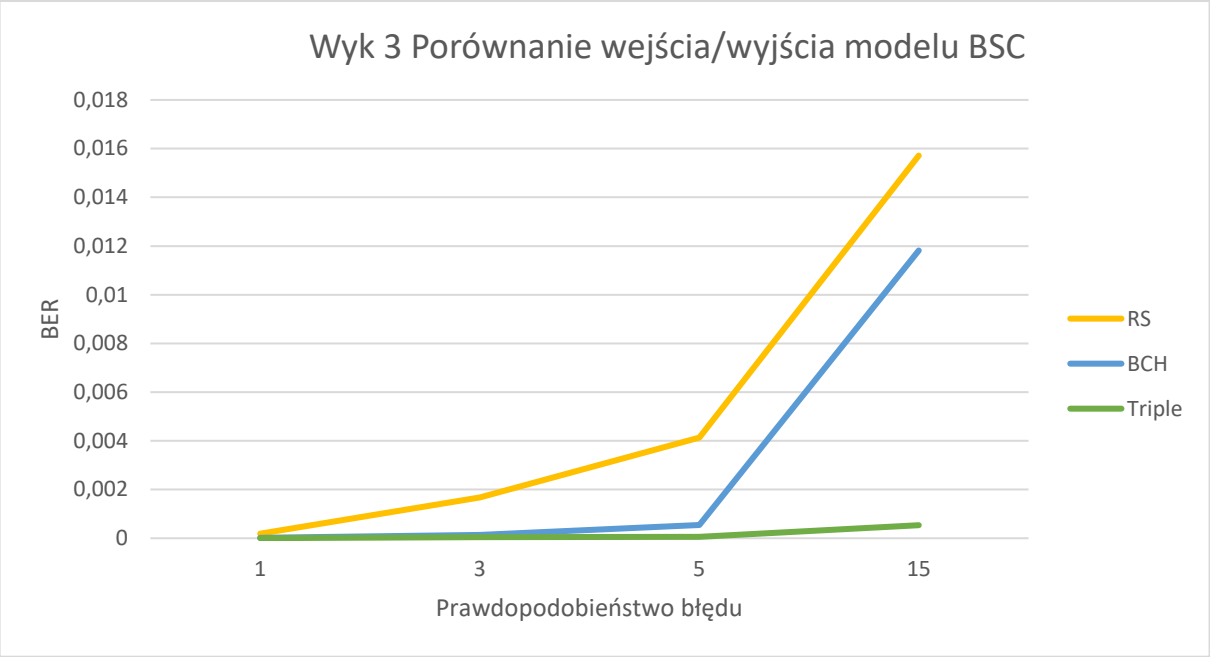
5.1 Wykres 1 N=255 K=239 T=2 model BSC



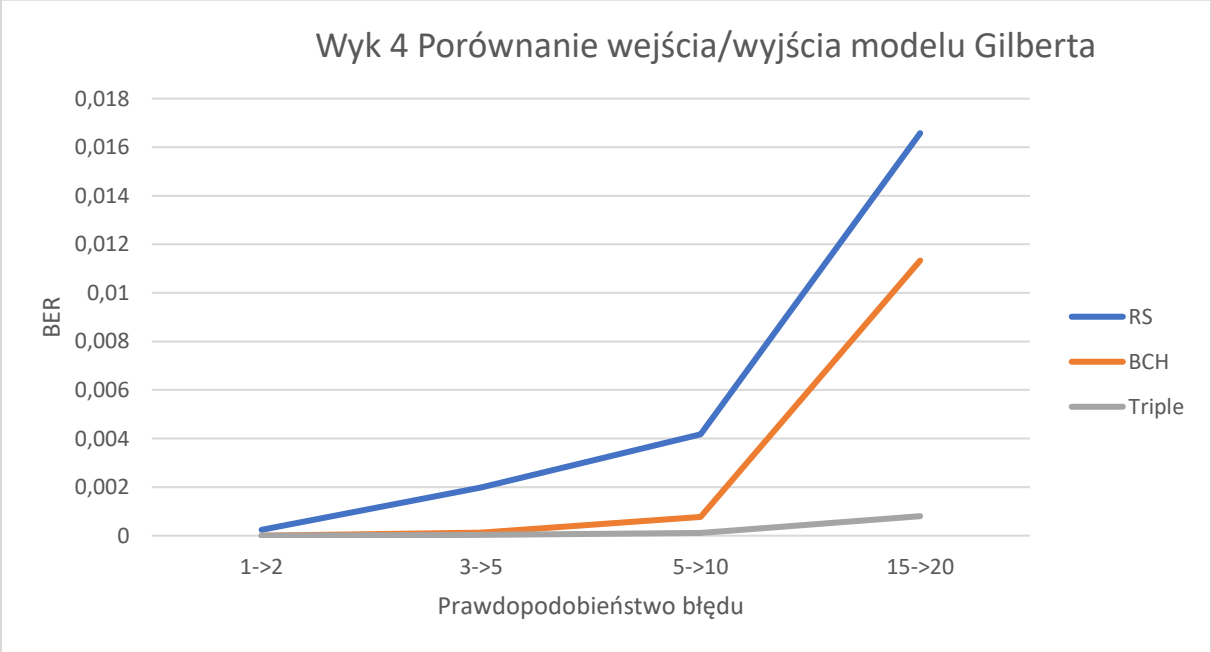
5.2 Wykres 2 N=255 K=239 T=2 model Gilberta



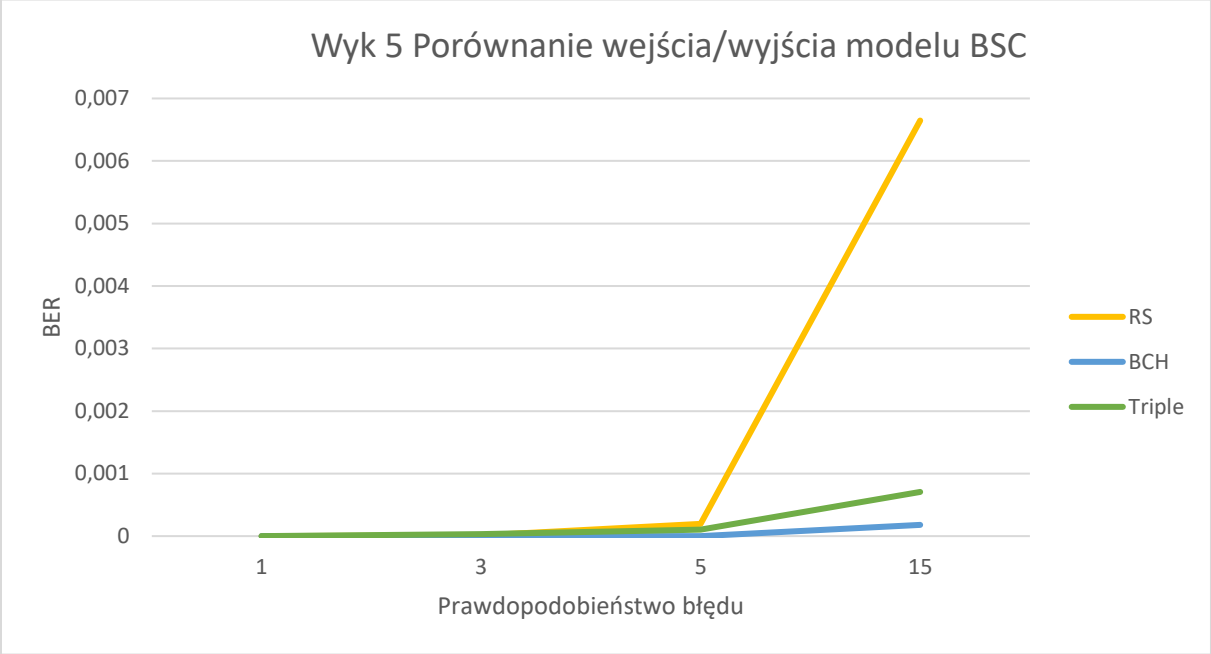
5.3 Wykres 3 N=255 K=231 T=3 model BSC



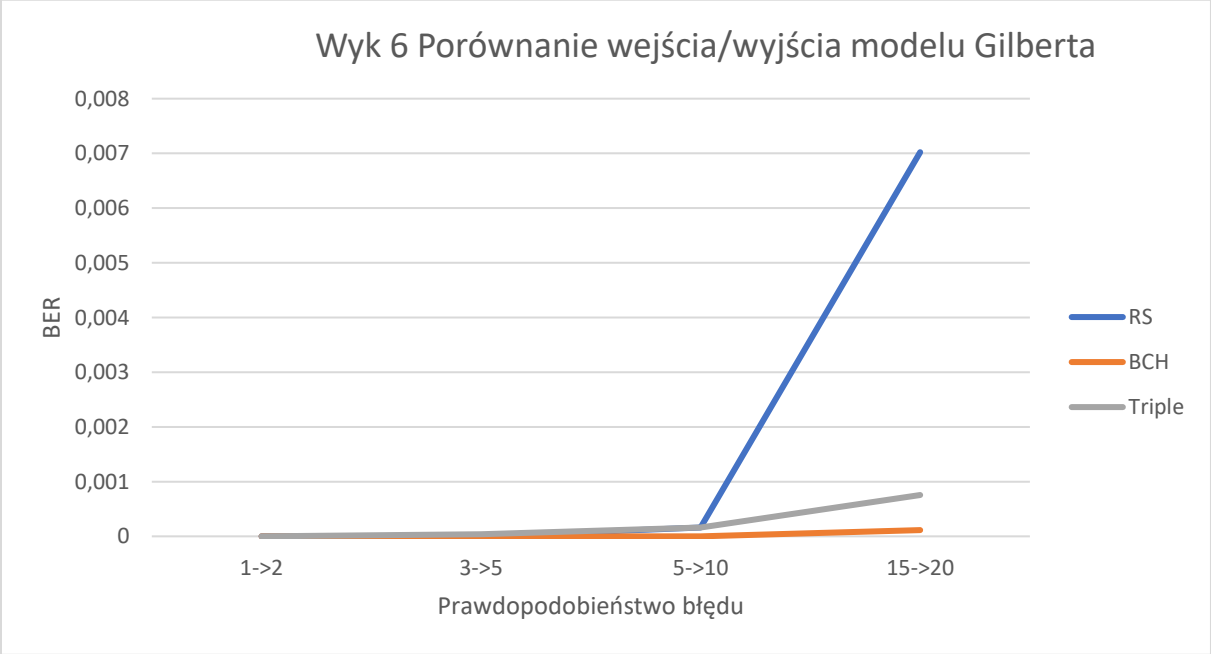
5.4 Wykres 4 Wykres 3 N=255 K=231 T=3 model Gilberta



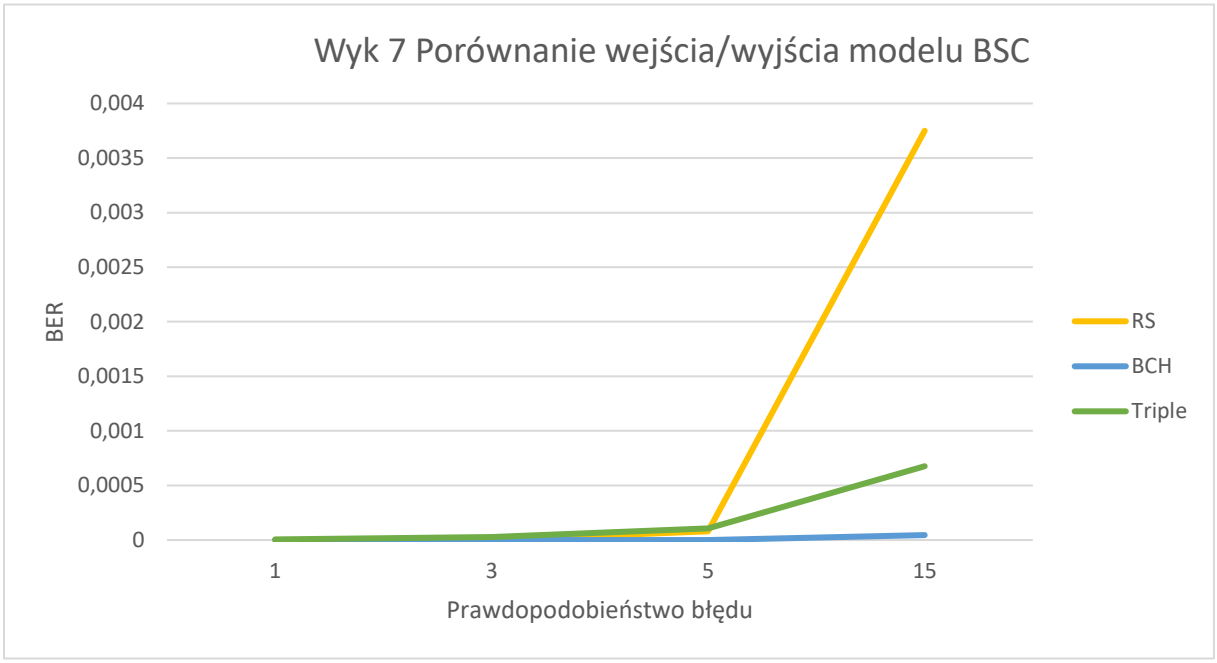
5.5 Wykres 5 N=255 K=187 T=9 model BSC



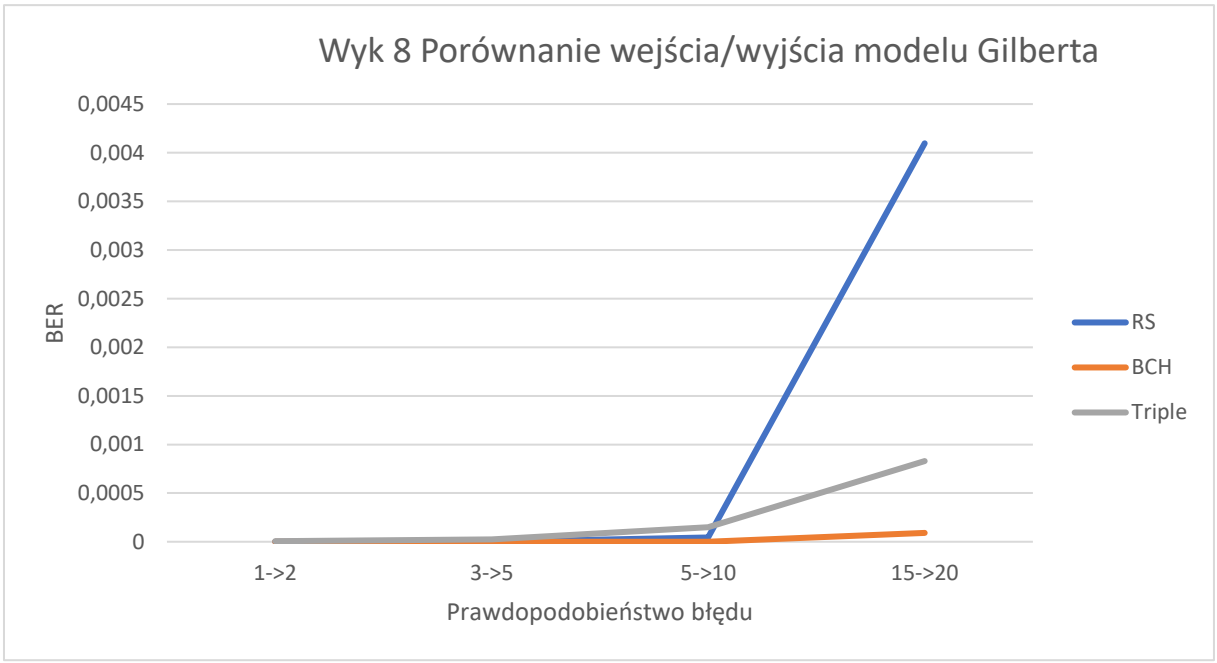
5.6 Wykres 6 N=255 K=187 T=9 model Gilberta



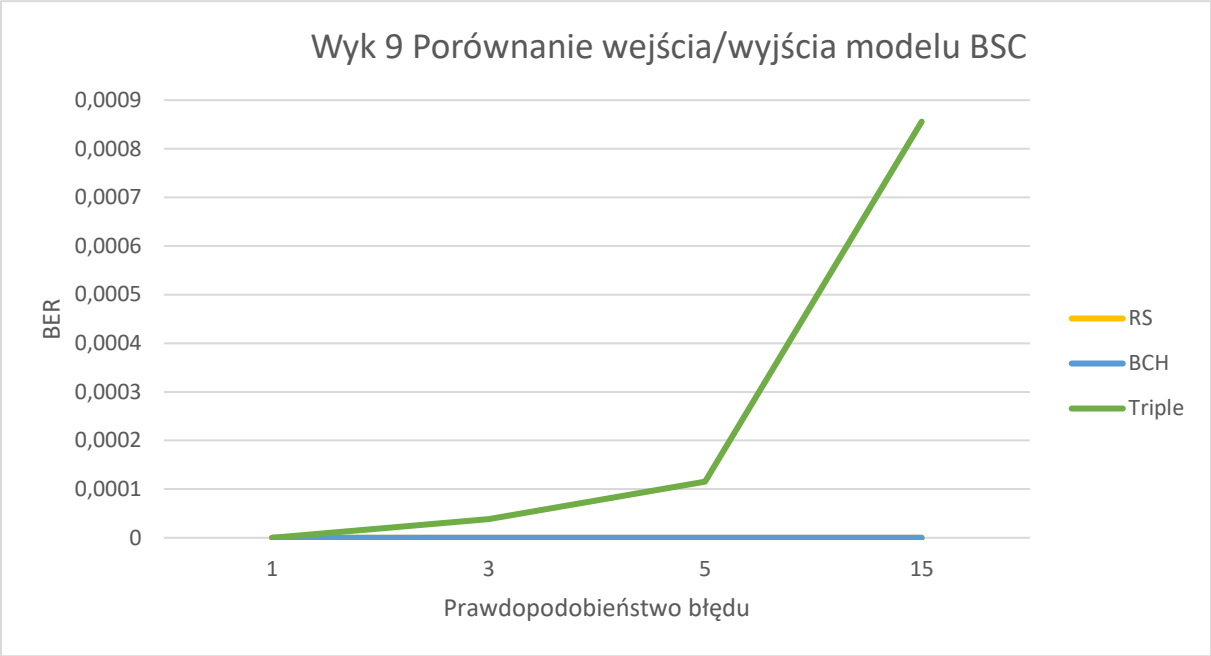
5.7 Wykres 7 N=255 K=179 T=10 model BSC



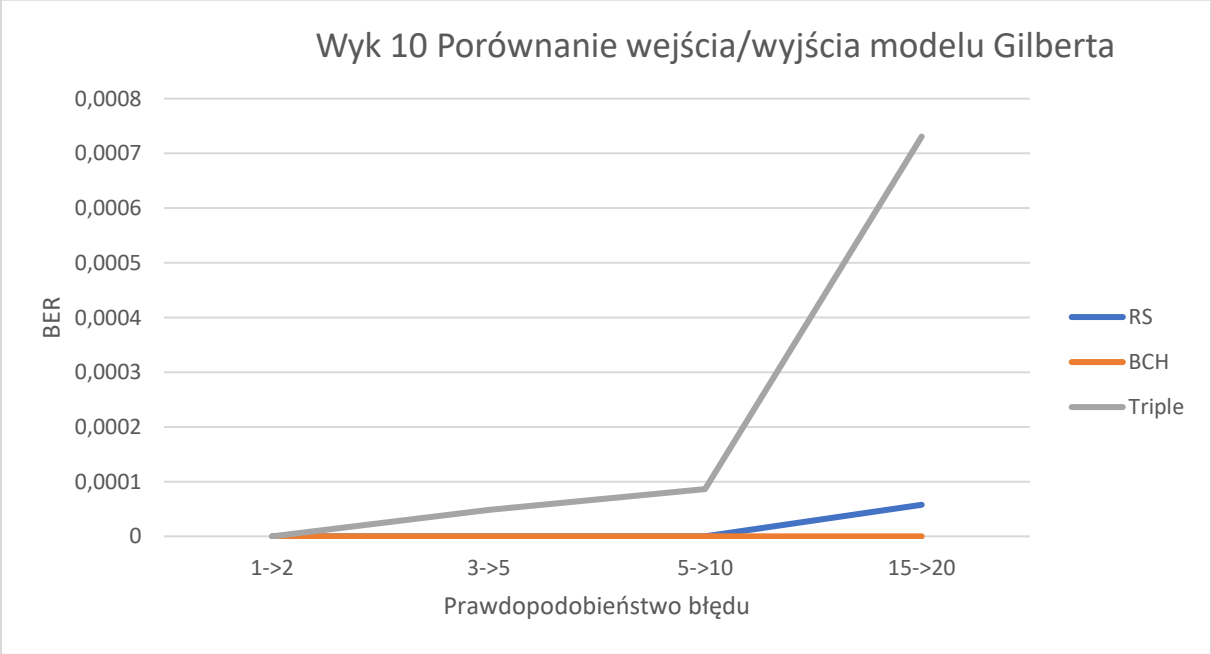
5.8 Wykres 8 N=255 K=179 T=10 model Gilberta



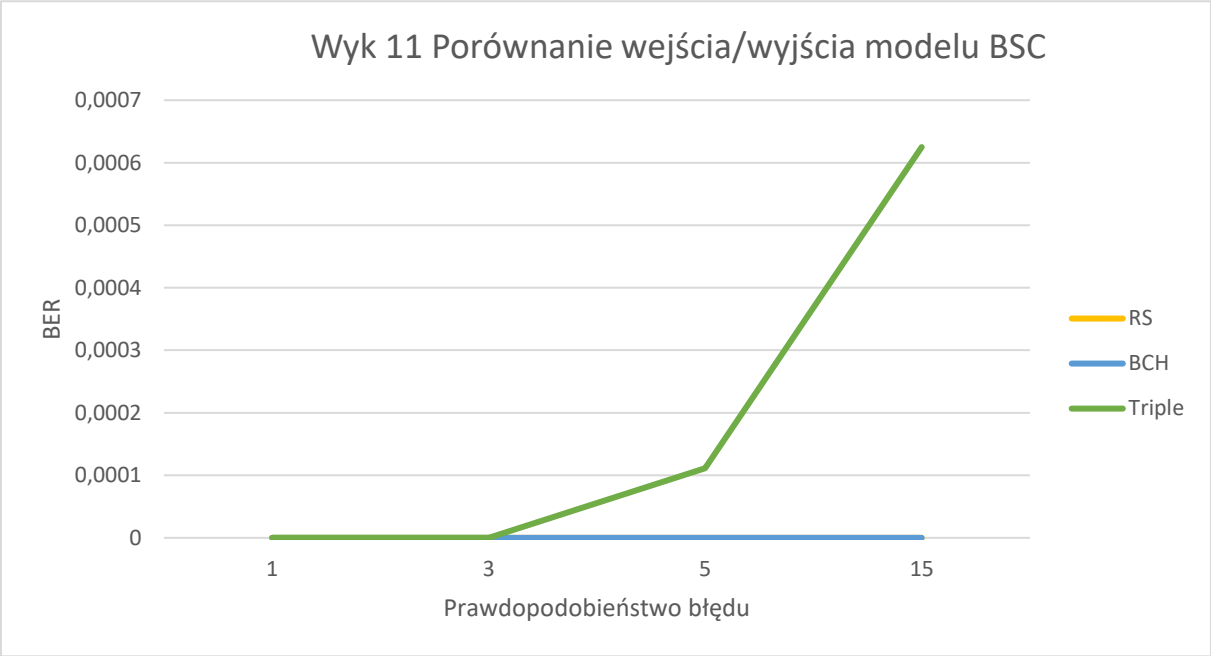
5.9 Wykres 9 N=255 K=107 T=22 model BSC



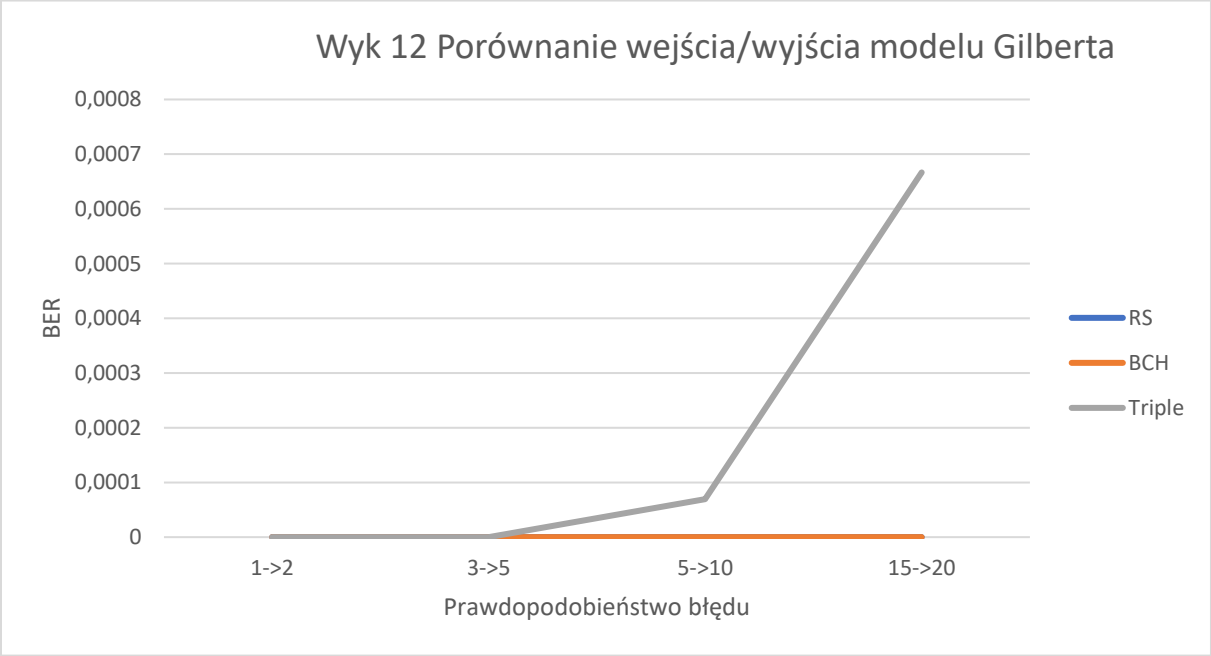
5.10 Wykres 10 N=255 K=107 T=22 model Gilberta



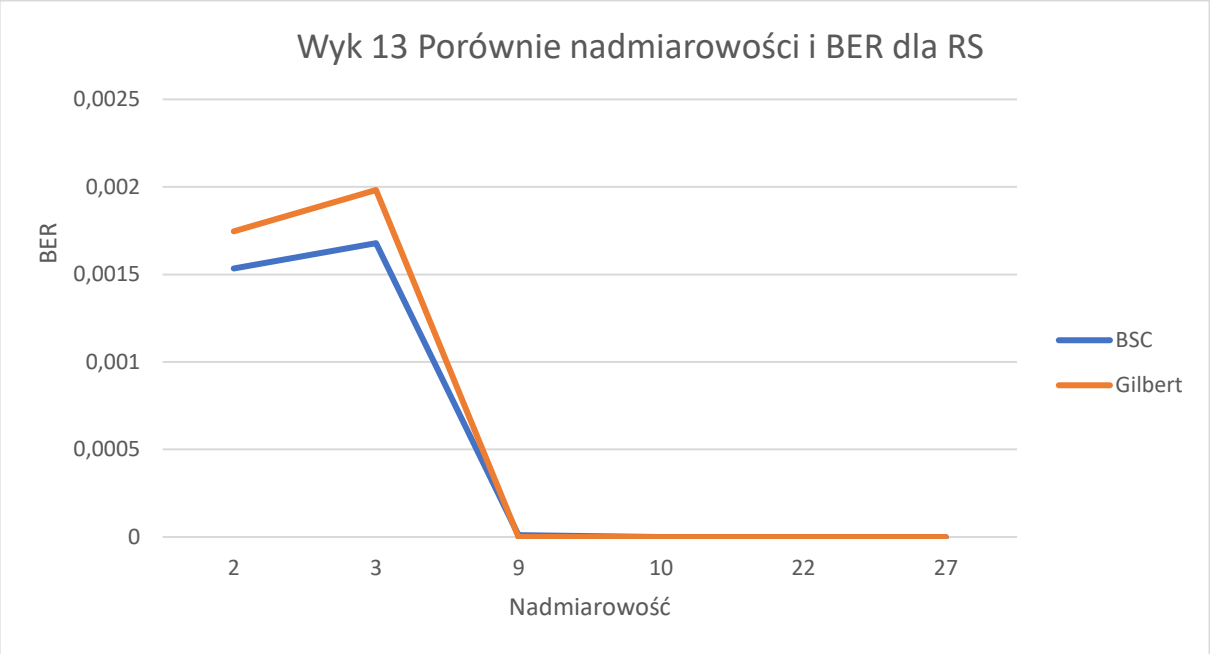
5.11 Wykres 11 N=255 K=79 T=27 model BSC



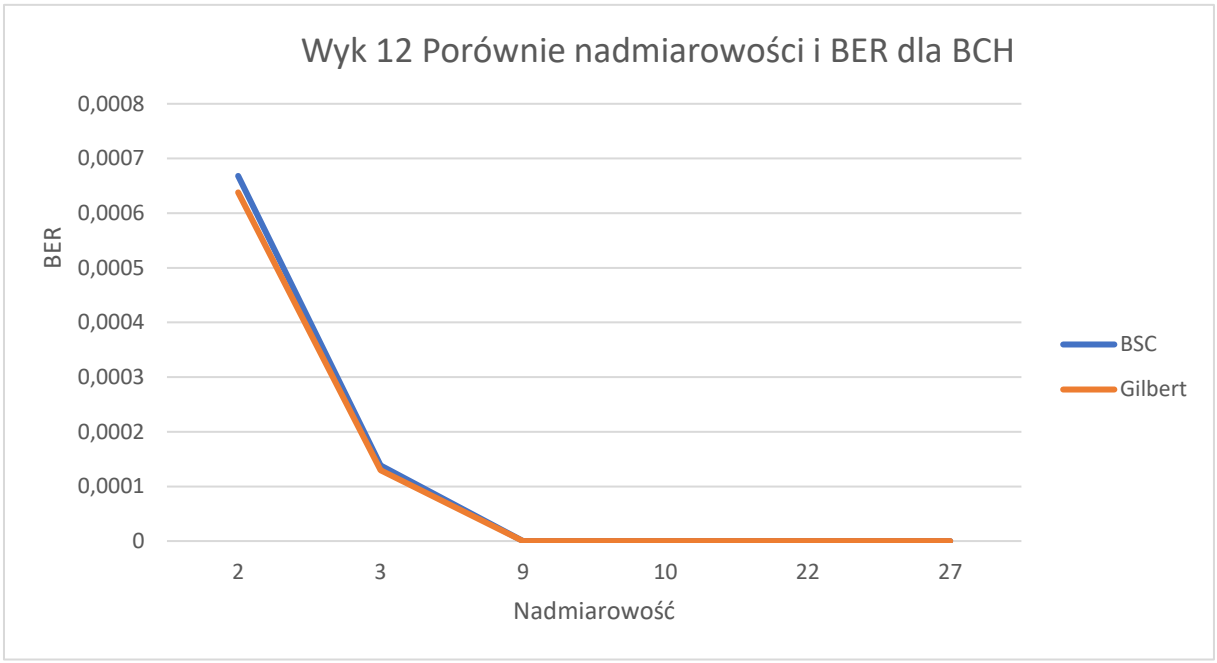
5.12 Wykres 12 N=255 K=79 T=27 model Gilberta



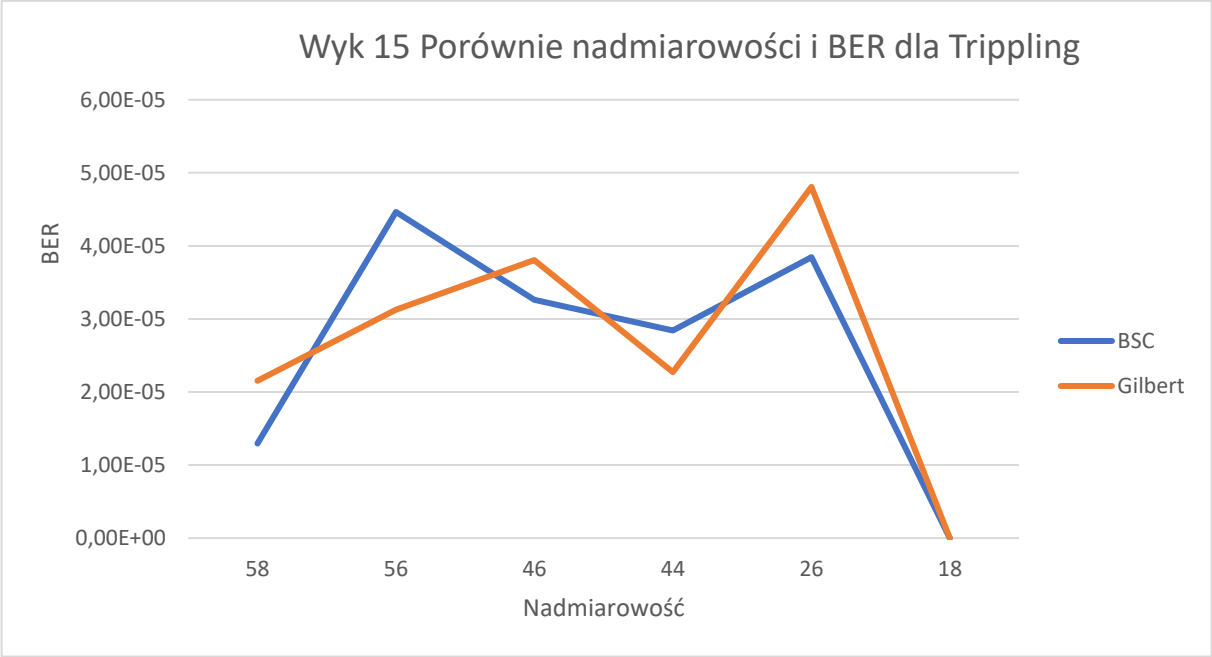
5.13 Wykres 13 Porównanie nadmiarowości i BER dla RS



5.14 Wykres 14 Porównanie nadmiarowości i BER dla BCH



5.15 Wykres 15 Porównanie nadmiarowości i BER dla Tripling



Wartości na osi x maleją, ponieważ maleje długość słowa wysyłanego.

6 Wnioski

Według wykresów od 1 do 8 widać, że kod potrojeniowy ma znacznie lepszą korekcje błędów od pozostałych koderów. Jednak gdy spojrzymy na wykresy 13-15 widać, że wynika to jedynie z tego, że kod potrojeniowy ma znacznie więcej bitów nadmiarowych. Na wykresach 13 i 14 widać, że BER znacznie maleje, jeżeli nadmiarowość rośnie. Już przy nadmiarowości 9 dla przedstawionych kodów wartość BER jest bliska 0, jednak dla kodu potrojeniowego wartość ta prawie nigdy nie jest równa 0.

Wynika to z tego, że dla kodów RS i BCH korekcja nie wymaga pewności, że dwa bity obok siebie nie zostały zmienione. Oznacza to, że kod potrojeniowy nie sprawdzi się szczególnie w sytuacjach, gdy błędy seryjne są częstsze (model kanału Gilberta).

Na wykresach widać również, że wartość BER dla kodów RS szczególnie rośnie przy większej ilości błędów losowych. Może wynikać to z tego, że koder nie jest w stanie już nic zdekodować, gdy przekroczymy pewną wartości ilości błędów, lub bity parzystości zostaną za bardzo zniekształcone.

Co nasuwa się już z samego założenia, na wykresach widać, że zwiększając ilość bitów parzystości, zmniejsza nam się BER sygnału odebranego. Jednak podczas projektowania systemu należy pamiętać o koszcie nadmiarowości kodu. Najlepiej założyć na początku wymaganie co do systemu, przy jakim modelu kanału transmisyjnego, jakie jest prawdopodobieństwo zakłamania kodu. Im większy nadmiar, tym mniej danych można wysłać w jednym segmencie, co zmniejsza przepustowość kanału dla danych typu goodput (danych użytecznych).

Dla kodu potrojeniowego nie ma możliwości dobierania takich parametrów, co sprawia, że jest on mocno nieprzydatny. Jednak podczas projektowania systemu z użyciem kodera BCH należy pamiętać o odpowiednim doborze parametrów wielomianu generującego i słowa kodowego.