Symulacja i Analiza Systemu Transmisji FEC

Prowadzący: dr hab. inż. Henryk Maciejewski

Skład grupy: Aliaksei Tokarau 250930, Maciej Tylak 248884

Wprowadzenie

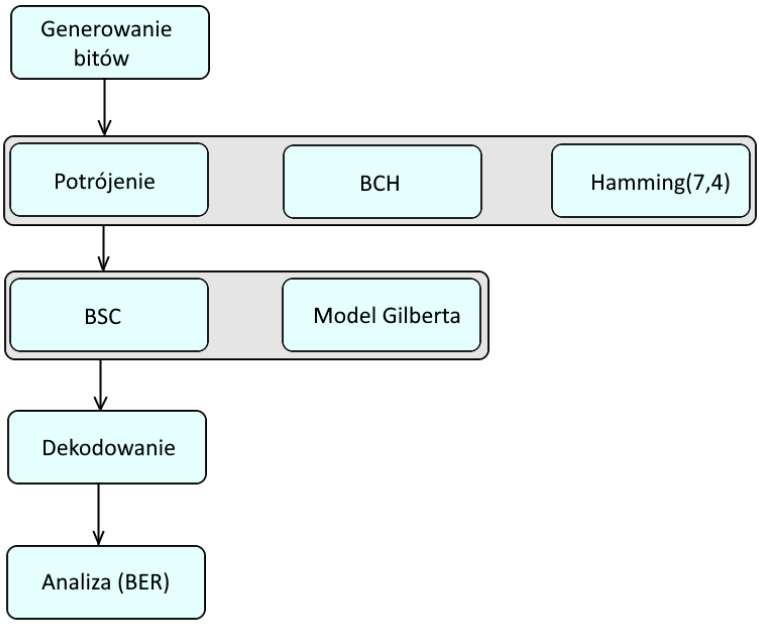
Forward error correction (FEC) jest techniką korekcji błędów służącą do wykrywania i korygowania ograniczonej liczby błędów w przesyłanych danych bez retransmisji. W tej metodzie nadawca wysyła redundantny kod korygujący błąd wraz z ramką danych. Odbiornik wykonuje niezbędne kontrole na podstawie dodatkowych nadmiarowych bitów. Jeśli stwierdzi, że dane zawierają błędy, wykonuje kod korygujący błędy, który generuje rzeczywistą ramkę. Następnie usuwa nadmiarowe bity przed przekazaniem wiadomości do górnych warstw.

Wady i Zalety FEC:

- Ponieważ FEC nie wymaga handshakingu między źródłem a celem, może być używany do transmisji danych do wielu miejsc przeznaczenia jednocześnie z jednego źródła.
- Kolejną zaletą jest to, że FEC oszczędza pasmo wymagane do retransmisji. Jest więc używany w systemach czasu rzeczywistego.
- Jego głównym ograniczeniem jest to, że jeśli jest zbyt wiele błędów, ramki muszą być retransmitowane.
- Wadą są również dość duże nadmierności kodu – dużo bitów poświęcone jest na korekcję błędów.

Opis systemu:

Program został wykonany całkowicie w środowisku Matlab. Najpierw generowany jest ciąg bitów za pomocą pseudolosowej funkcji. Bity generowane były na pięć sposobów: potrojenie, Hamminga(7,4) i BCH – w trzech wariacjach (BCH(15,5), BCH(511,502), BCH(511,130). Zrealizowane zostały dwa kanały: binarny kanał symetryczny (BSC) oraz model Gilberta, przez które transportowane zostały zakodowane bity. Na końcu po zdekodowaniu przeanalizowaliśmy różne kombinacje kodowań oraz kanałów transmisyjnych pod kątem współczynnika błędów BER oraz nadmiarowości kodu.



Opis etapów symulacji

1. **Generowanie bitów**

   Zmienna **n_bits** odpowiada za ilość wygenerowanych bitów dla uruchomienia symulacji. Chcemy wygenerować tablicę, a nie macierz, więc ustawiamy zmienną **rows** (wierszy) na jedynkę. Dalej zapisujemy bity do tablicy **init_bits** za pomocą wbudowanej funkcji **randi(_)** która zwraca pseudolosowe liczby wykorzystując rozkład jednostajny ciągły. Jako argumenty podajemy zakres liczb [0, 1], rows, czyli ilość wierszy oraz ilość bitów.

2. **Kodowanie**

   a) **Potrójenie**
   Wykorzystujemy wbudowaną funkcje **repelem(_)** która powtarza każdy bit **n** razy w wektorze **v**. Jako wektor podajemy wygenerowaną tablice bitów **init_bits**, n jest trójką bo chcemy potroić bity.

   b) **Bose–Chaudhuri–Hocquenghem (BCH)**
   Wybieramy prostą liczbę **n** taką aby $d \leq q^m - 1$, gdzie **m** i **d** są liczbami naturalnymi. Dalej wyznaczamy długość wiadomości **k**. Liczby n i k muszą tworzyć kod BCH. Następnym krokiem jest przetwarzanie pierwotnych bitów do postaci macierzy **n_bits/k x k** aby możliwe było wygenerować tablicę ciała skończonego za pomocą funkcji **gf(_)**. Dalej stosujemy wbudowaną funkcję **bchenc(_)** i otrzymujemy zakodowane BCH bity. Jako argumenty podajemy tablicę ciała skończonego **msgTx**, prostą liczbę **n** oraz długość wiadomości **k**.

c) **Kod Hamminga**
Wbudowana funkcja **encode(_)** pozwala na kodowanie danych wybraną metodą. W naszym przypadku to **„hamming/binary”**. Długość słowa kodowego: 7, długość wiadomości: 4.

3. **Transmisja**

a) **Binarny kanał symetryczny (BSC)**
Ustawiamy prawdopodobność wystąpienia błędu **probability**. Dalej zapisujemy transportowane dane do tablicy **ndata** przy tym zmieniając ich za pomocą funkcji **bsc(_)**.

b) **Kanał Gilberta**
Zmienna **goodtobad** - prawdopodobieństwo przejścia ze stanu dobrego do złego,
badtogood - prawdopodobieństwo przejścia ze stanu dobrego do złego,
errorwhengood - prawdopodobieństwo błędu, gdy w stanie dobrym ,
errorwhenbad - prawdopodobieństwo błędu, gdy w stanie złym.
Dodatkowo, jeżeli transportujemy bity przez kanał Gilberta i wykorzystujemy BCH, mamy przetworzyć dane na przejrzystą formę, bo po zakodowaniu do BCH są one w postaci wielomianowej. Do tego definiujemy funkcje **gf2dec(_)**, która przetwarza tablicę ciała skończonego do postaci dziesiętnej. Też po przetworzeniu danych w kanale Gilberta wracamy do postaci wielomianowej.

4. **Dekodowanie**

a) **Potrójenie**
W cykle liczymy wystąpienia zer i jedynek w otrzymanym kodzie. Gdy ilość liczb dochodzi do trzech, porównujemy ilość zer (**zeros**) i jedynek (**ones**). Jeżeli jedynek jest więcej niż zer to kolejnym elementem tablicy zdekodowanej staje się jedynka, W przeciwnym przypadku – zero.

b) **Bose–Chaudhuri–Hocquenghem (BCH)**
Dekodujemy otrzymane dane za pomocą funkcji **bchenc(_)** i zapisujemy rezultat do tablicy **bch_encoded**, parametry funkcji są opisane wyżej. Dalej przetwarzamy bity z postaci wielomianowej do postaci dziesiętnej za pomocą zdefiniowanej funkcji **gf2dec(_)**.

c) **Kod Hamminga**
Stosujemy wbudowaną funkcję **decode(_)**, która dekoduje dane które były zakodowane wybraną metodą. Parametry są opisane wyżej.

5. **Analiza (BER) i redundancji**

W niezależności od metody liczymy liczbę i częstotliwość wystąpienia błędów bitowych stosując wbudowaną funkcję **biterr(_)**, jako argumenty podajemy otrzymane dane (**triple_BER, hamming_BER, bch_BER**. Stosunek Ilości bitów otrzymanych i pierwotnych jest redundancją, czyli nadmiarowością kodu (**triple_redundancy, hamming_redundancy, bch_redundancy**).

Teoria

- Kod potrojony – prosty i prymitywny sposób kodowania, gdzie każdy bit w wysyłanej wiadomości jest potrajany (np. zakodowana wiadomość „010” ma postać „000111000”). Przy dekodowaniu porównywane są 3 kolejne bity i dominująca wartość zostaje podana przy dekodowaniu (np. 001 dekodowane jest na 0, a 110 na 1). Nadmiarowość kodu wynosi 3.

- Kod Hamminga – opracowany przez Richarda Hamminga, kod ten polega na nadaniu w danej wiadomości bitów parzystości (pozycje będące potęgami dwójki) i bitów informacyjnych (pozostałe). Bity parzystości określają, które bity powinny być sprawdzane, a które nie. Dodatkowo w kodzie umieszczane są bity parzystości, mające na celu wykrycie błędów transmisji. Kod Hamminga(7,4) jest najpopularniejszą implementacją kodu Hamminga – zapewnia ona 4 bity danych w siedmio-bitowych blokach, razem z trzema bitami parzystości. Nadmiarowość kodu to około 1.75 .

- Kod BCH(n,k) – nazwa pochodzi od nazwisk odkrywców kodu (Bose–Chaudhuri–Hocquenghem). Kod wykorzystuje wielomian generujący (stopnia n-k) niezbędny do utworzenia słowa kodowego. Ciąg informacji przesunięty jest w lewo o n-k pozycji i następnie mnożony przez wielomian generujący. Możliwe są tylko konkretne wartości n i k, a nadmiarowość kodu to ich iloczyn.

- Kanał BSC – prosty model kanału przepływu danych. Zakodowane bity w realnym świecie ulegają błędom transmisji, a BSC jest prostym w zrozumieniu modelem. Istnieje prawdopodobieństwo *p*, że dany bit podczas transmisji zostanie przesłany z błędem (0 na 1, 1 na 0) – *p* jest jedynym parametrem kanału.

- Model Gilberta – Elliotta – model kanału, który może być w dobrym lub złym stanie. Jeśli jest w dobrym stanie, dane przepływające przez niego poddane są bardzo niskiemu prawdopodobieństwu wystąpienia błędu. Jeśli natomiast jest w złym stanie, szansa na błąd jest duża. Przechodzenie między stanami również jest obarczone swoimi zasadami – model jest w stanie przejść ze stanu dobrego do złego i vice versa w trakcie swojej pracy – zmiana stanu obarczona jest jakimś prawdopodobieństwem (zazwyczaj modele konstruowane są tak, aby przejście ze stanu dobrego do złego występowała rzadko, a ze złego do dobrego często). Kanał modelowany jest na podstawie procesu Markova, gdzie prawdopodobieństwo jednego zdarzenia jest zależne jedynie od wyniku poprzedniego.

Organizacja eksperymentu

Eksperyment polegał na przeprowadzeniu pełnej symulacji w różnych konfiguracjach kodowań i kanałów. Po zakończeniu transmisji przedstawiane zostały obliczone wartości **BER** oraz **redundancja** dla każdego modelu przepływu danych.

- Każda symulacja w danym ustawieniu kodowań i kanałów została powtórzona 10 razy, a wartość BER została uśredniona, dzięki czemu otrzymaliśmy bardziej wiarygodne wyniki
- Ilość wysłanych bitów n_bits = 100 000 – zauważyliśmy, iż dla ilości większej od 100 000 różnica obliczanego BER nie różniła się w zauważalnym stopniu, a mniejsza ilość bitów skraca czas wymagany na przeprowadzenie symulacji.

Użyte kodowania:

- Potrojenie
- Hamming(7,4)
- BCH(15,5)
- BCH(511,502)
- BCH(511,130)

Parametry kodowania BCH(n,k) zostały dobrane w taki sposób, aby sprawdzić różne redundancje, jak również długości wielomianów i ich wpływ na otrzymaną wartość BER.

Użyte kanały i ich parametry:

- BSC

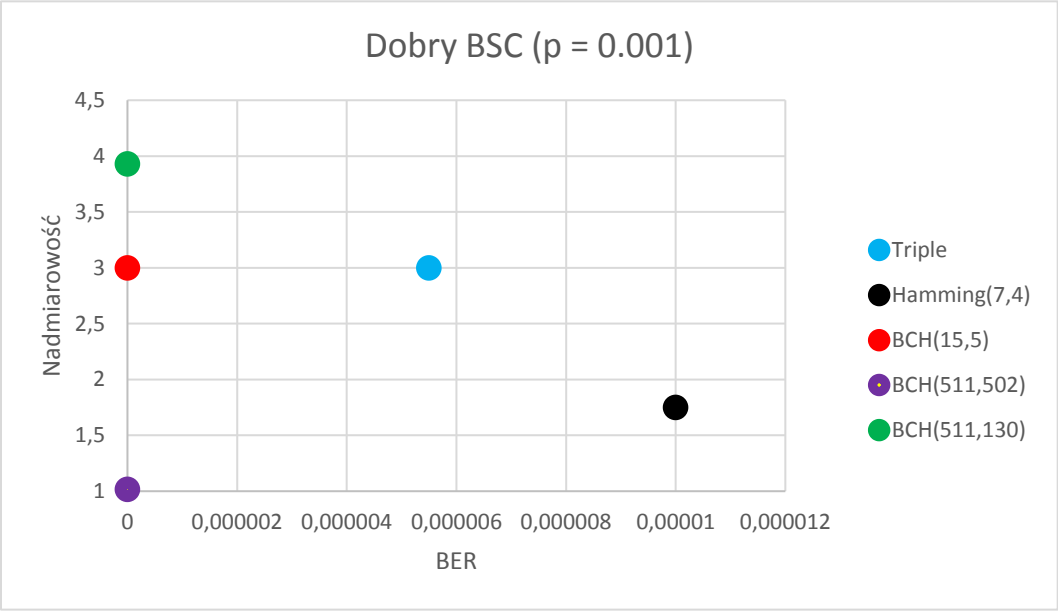
Szanse wystąpienia błędu = 0.001, 0.05, 0.4

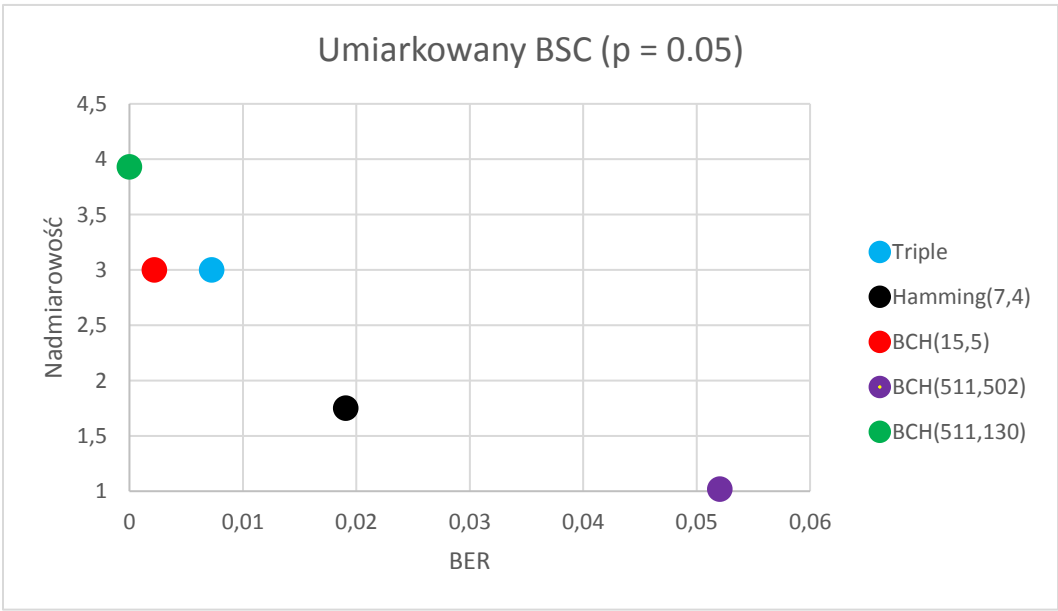
- Model Gilberta – Elliotta

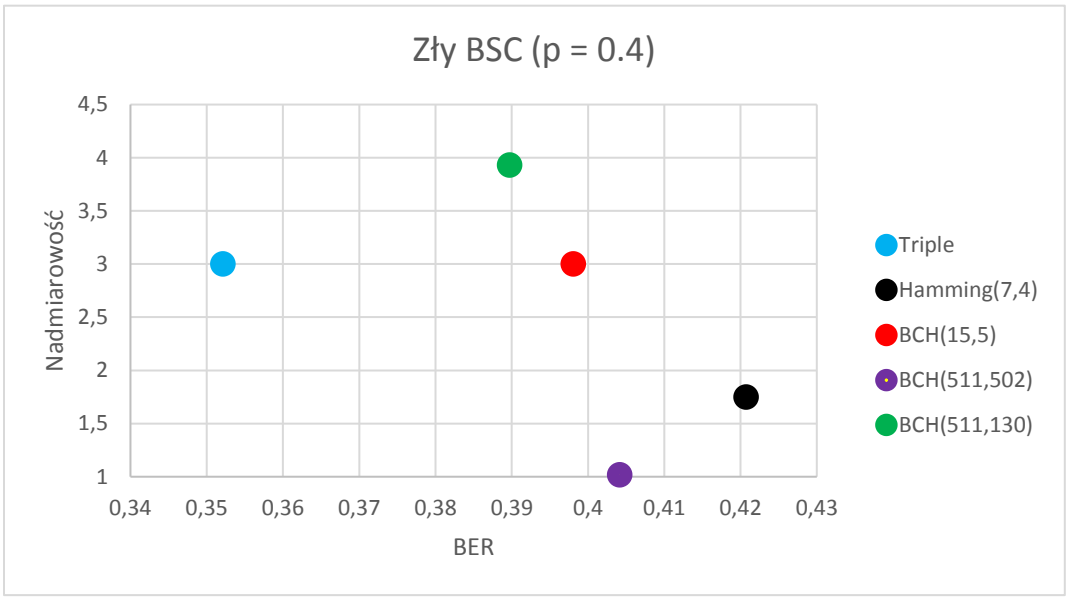
Parametry (goodtobad, badtogood, errorwhengood, errorwhenbad):
(0.0001, 0.8, 0.00005, 0.4)
(0.004, 0.2, 0.0005, 0.9)
(0.005, 0.005, 0.0005, 0.995)

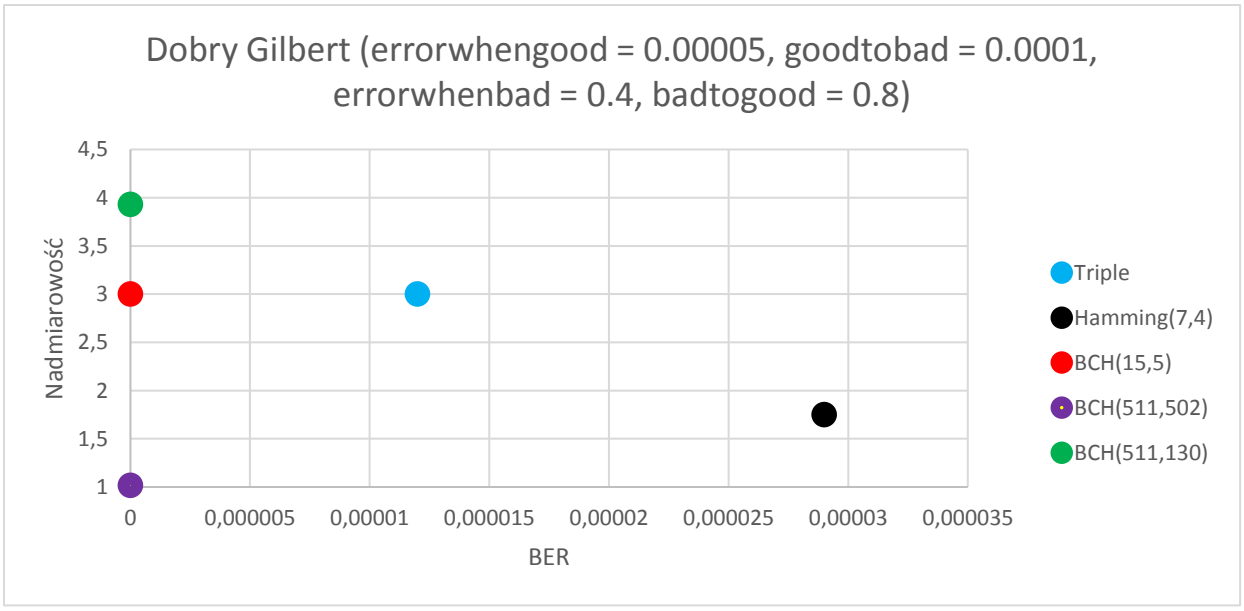
Wykorzystane modele kanałów gwarantują nam zbadanie efektywności kodowań w różnych sytuacjach. Podzieliliśmy kanały pod kątem jakości transmisji na trzy kategorie – dobry, umiarkowany, zły. O wiele łatwiej jest ustawić parametry w kanale BSC (jest to zwykła szansa na niepoprawne przesłanie bitu), niż w przypadku modelu Gilberta – Elliotta, jednak próbowaliśmy ustalić parametry tak, aby BER był zbliżony dla poszczególnych jakości.

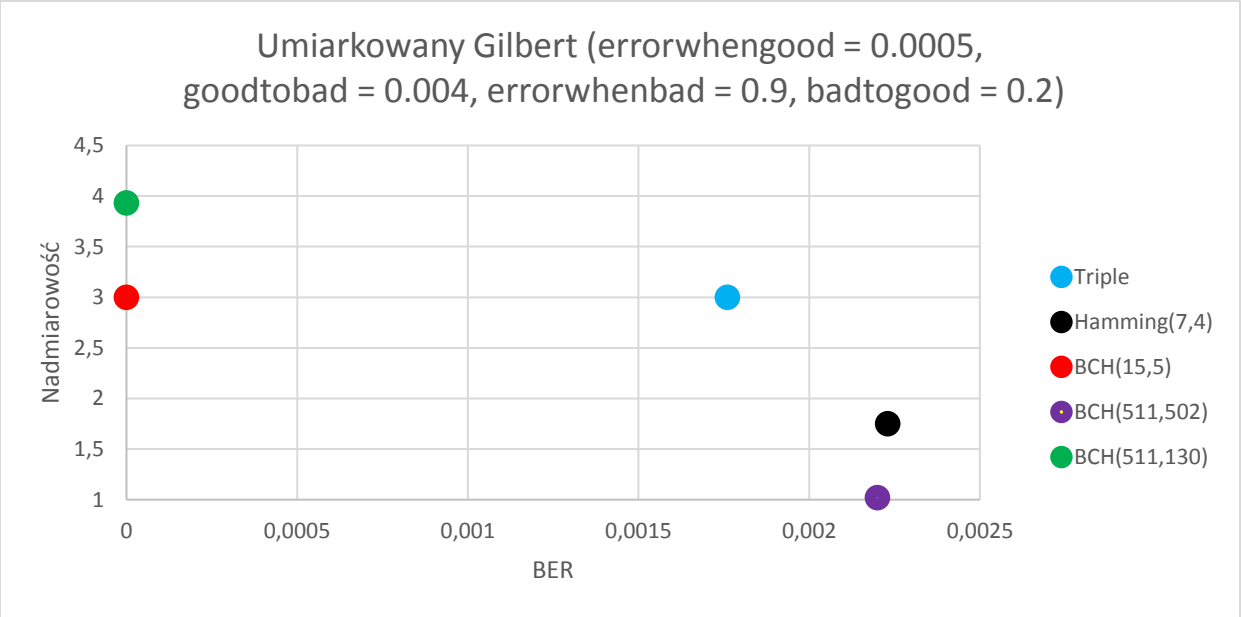
Wyniki

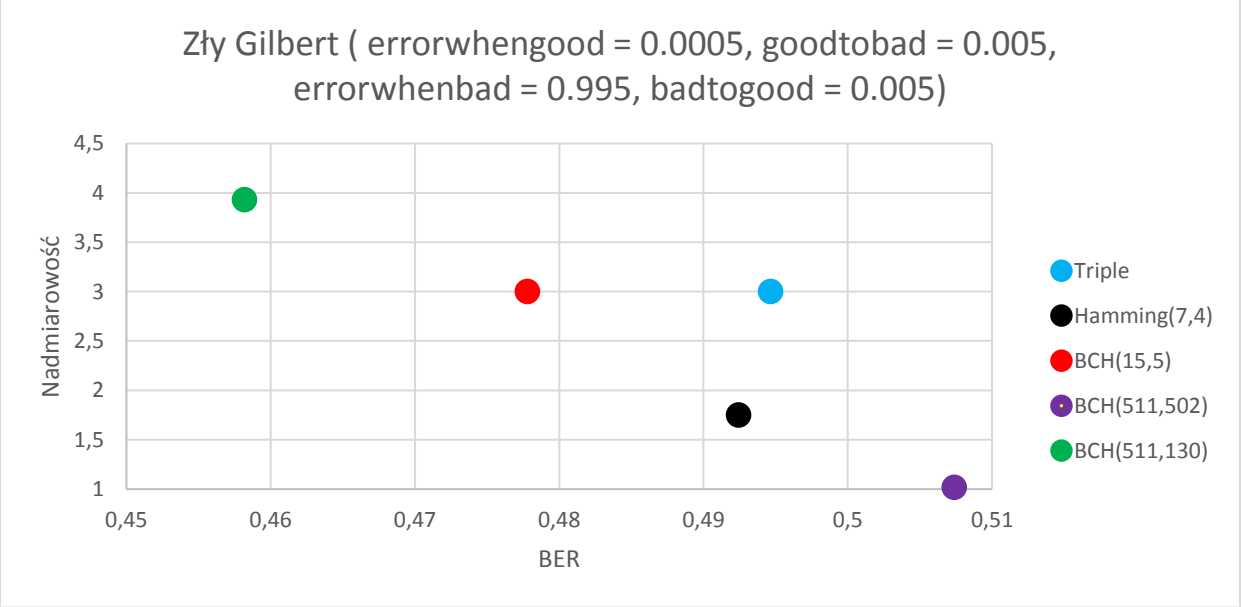

Dobry BSC (p = 0.001)
Nadmiarowość
4,5
4
3,5
3
2,5
2
1,5
1
0
0,000002
0,000004
0,000006
0,000008
0,00001
0,000012
BER
Triple
Hamming(7,4)
BCH(15,5)
BCH(511,502)
BCH(511,130)


Umiarkowany BSC (p = 0.05)
Nadmiarowość
4,5
4
3,5
3
2,5
2
1,5
1
0
0,01
0,02
0,03
0,04
0,05
0,06
BER
Triple
Hamming(7,4)
BCH(15,5)
BCH(511,502)
BCH(511,130)


Zły BSC (p = 0.4)
Nadmiarowość
4,5
4
3,5
3
2,5
2
1,5
1
0,34
0,35
0,36
0,37
0,38
0,39
0,4
0,41
0,42
0,43
BER
Triple
Hamming(7,4)
BCH(15,5)
BCH(511,502)
BCH(511,130)


Dobry Gilbert (errorwhengood = 0.00005, goodtobad = 0.0001, errorwhenbad = 0.4, badtogood = 0.8)
Nadmiarowość
4,5
4
3,5
3
2,5
2
1,5
1
0
0,000005
0,00001
0,000015
0,00002
0,000025
0,00003
0,000035
BER
Triple
Hamming(7,4)
BCH(15,5)
BCH(511,502)
BCH(511,130)


Umiarkowany Gilbert (errorwhengood = 0.0005, goodtobad = 0.004, errorwhenbad = 0.9, badtogood = 0.2)
Nadmiarowość
4,5
4
3,5
3
2,5
2
1,5
1
0
0,0005
0,001
0,0015
0,002
0,0025
BER
Triple
Hamming(7,4)
BCH(15,5)
BCH(511,502)
BCH(511,130)


Zły Gilbert (errorwhengood = 0.0005, goodtobad = 0.005, errorwhenbad = 0.995, badtogood = 0.005)
Nadmiarowość
4,5
4
3,5
3
2,5
2
1,5
1
0,45
0,46
0,47
0,48
0,49
0,5
0,51
BER
Triple
Hamming(7,4)
BCH(15,5)
BCH(511,502)
BCH(511,130)

Uśrednione wyniki transmisji danych w każdej konfiguracji zostały pokazane na wykresie (korzystając z programu Microsoft Excel).
Oś X opisuje bit error rate, a oś Y nadmiarowość (redundancję).

Analiza wyników

Kanały BSC

- Kanał dobry – w przypadku kanału dobrego wszystkie kodowania BCH nie wykazały żadnego błędu, natomiast kod potrojony wykazały BER rzędu 10^{-6}, a Hamminga 10^{-5}. Biorąc pod uwagę taki sam wynik dla wszystkich kodowań BCH, najoptymalniejsze jest w kanale dobrym wybranie kodu BCH(511,502) ponieważ posiada on najniższą redundancję (dokładnie 1.0179)

- Kanał umiarkowany – w kanale umiarkowanym widać już zależności pomiędzy poszczególnymi kodowaniami BCH. BCH(511,130) posiada największą redundancję i gwarantuje zerowy BER, natomiast BCH(511,502) z niską redundancją powoduje stosunkowo duży BER. BCH(15,5) okazało się lepsze od potrojonego (mimo tej samej redundancji). W tym kanale przy wyborze optymalnego sposobu należy zwrócić uwagę na potrzeby – należy wybrać odpowiedni próg tolerancji BER i pod tym kątem wybrać kodowanie z jak najmniejszą redundancją.

- Kanał zły – w tym kanale najefektywniejsze okazało się kodowanie potrojone. Wynika to z faktu, iż model takiego kanału nie nadaje się na kodowanie BCH – prawdopodobieństwo wystąpienia błędu jak za duże, a kodowanie ma mały wpływ na zaistniały BER. BCH natomiast utrzymało wcześniej zauważalne zależności (im większa redundancja BCH, tym mniejszy BER).

Model Gilberta – Elliotta

- Kanał dobry – wyniki są podobne do tych uzyskanych w kanale BSC – kodowania BCH gwarantują niezawodność transmisji i należy wybrać ten posiadający najmniejszą redundancję. Kod potrojony również okazał się lepszy od kodu Hamminga.

- Kanał umiarkowany – analiza wyników również wygląda podobnie jak w BSC. Widać już różnicę pomiędzy poszczególnymi kodowaniami BCH. W tym konkretnym modelu kodowanie BCH(15,5) jest optymalniejsze od BCH(511,130) – obydwa gwarantują brak wystąpień błędów. Kodowanie potrójne nadal jest lepsze od Hamminga, ale to z kolei jest gorsze od BCH(511,502). Wynika to oczywiście z faktu, iż jakość kanałów nie jest sobie równa – umiarkowany kanał BSC w naszym eksperymencie nie jest równy umiarkowanemu kanałowi modelu Gilberta – Elliotta i przy zmianie parametrów wyniki wyglądałyby podobnie jak w modelu BSC.

- Kanał zły – BER wszystkich kodowań oscyluje wokół 0.5, jednak zauważalna jest różnica w kodowaniach BCH (im większa redundancja, tym mniejszy BER). Co ciekawe, w modelu Gilberta – Elliotta kodowanie potrójne nie jest już najlepszym rozwiązaniem w kanale z bardzo złą jakością transmisji.

Analizując wszystkie wykresy jesteśmy w stanie zauważyć pewne trendy. Kod BCH jest zdecydowanie najefektywniejszym sposobem kodowania danych w naszych modelach – odpowiednie parametry gwarantują niską redundancję lub niski BER, a zależnie od potrzeb jesteśmy w stanie łatwo obliczyć najoptymalniejszą kombinację parametrów n,k dla BCH(n,k). Największą zaletą kodu Hamminga jest niska redundancja, jednak w badaniach okazuje się mniej efektywny od kodu BCH, a w niektórych sytuacjach optymalniejsze jest nawet rozważenie kodowania potrojonego.

Wnioski

Jest to teoretyczna implementacja FEC. W rzeczywistości implementacja FEC trochę się różni w zależności od zastosowania. Wynika to z faktu, iż utrata pakietów ma charakter losowy i występuje ze względu na wybuchy ruchu w okresach intensywnego użytkowania sieci. W rezultacie utrata pakietów zmienia się znacznie w czasie, nawet w okresie krótszym niż 24 godziny. Optymalna implementacja zmienia ilość pakietów FEC w celu uwzględnienia różnych warunków strat. Innymi słowy, więcej pakietów FEC będzie stracono w okresie wysokiej utraty pakietów sieciowych; mniej pakietów FEC będzie stosowany w okresach małej utraty pakietów. To pomaga zrównoważyć korzyści wydajności FEC oraz koszty ogólne, które rozwiązanie z natury wprowadza. Są różne firmy które wprowadzają różne rozwiązania dla FEC (np. Silver peak).

Generalnie FEC stosuje się w sytuacjach, gdy transmisja jest kosztowna lub niemożliwa, np. jednokierunkowe łącza komunikacyjne, oraz w przypadku transmisji do wielu odbiorników w technologii multicast. Informacje FEC są zazwyczaj dodawane do urządzeń pamięci masowej (magnetycznej, optycznej i półprzewodnikowej / flash) w celu umożliwienia odzyskiwania uszkodzonych danych, są szeroko stosowane w modemach, są stosowane w systemach, w których podstawową pamięcią jest pamięć ECC oraz w sytuacjach gzy stosujemy broadcast, w których odbiornik nie ma możliwości re-transmisji lub spowodowałoby to znaczne opóźnienie. Na przykład, w przypadku satelity orbitującego wokół Uranu, ponowna transmisja z powodu błędów dekodowania może spowodować opóźnienie o co najmniej 5 godzin.

Podaną symulację można stosować do testowania różnych sposobów kodowania danych, transmisji oraz do sprawdzenia ich wydajności i efektywności. Na danym symulatorze nie da się zasymulować rzeczywistego systemu, ale pozwala on na porównanie praktycznych i teoretycznych danych związanych z tymi sposobami kodowania i retransmisji. Można zauważyć również zależności między konkretnymi kodowaniami, opisane w poprzednim dziale.