

# Sprawozdanie z projektu

SYMULACJA SYSTEMU ARQ

FILIP OLSZEWSKI, KRZYSZTOF AGIEŃCZUK, JAKUB PERENC, KAJETAN  
PARZYSZEK

## Spis treści

Spis Ilustracji .....	2
1. Temat .....	3
2. Instrukcja obsługi .....	3
3. Opis implementacji kodu .....	4
a. Error Control .....	4
CRC32 (Cyclic Redundancy Check 32 bit) .....	4
2z5 .....	4
Parity bit (bit parzystości) .....	5
b. ARQ Protocols .....	5
Stop-and-Wait .....	5
Go-Back-N .....	5
c. Transfer Models .....	5
Binary Symmetric Channel (BSC) .....	5
Binary Erasure Channel (BEC) .....	5
Cyclic Error Channel (CEC) .....	6
4. Opis skryptu uśredniający wynik z wielu prób .....	6
5. Analiza Wyników: .....	7
a. Protokół SAW .....	7
Kanał BSC (na 1 osi 3 sposoby kontroli danych) .....	7
Kanał BEC (na 1 osi 3 sposoby kontroli danych) .....	8
Kanał CEC (na 1 osi 3 sposoby kontroli danych) .....	9
b. Protokół GBN .....	10
Kanał BSC (na 1 osi 3 sposoby kontroli danych) .....	10
Kanał BEC (na 1 osi 3 sposoby kontroli danych) .....	11
Kanał CEC (na 1 osi 3 sposoby kontroli danych) .....	12
c. Dodatkowo - działanie PB dla 2 różnych rozmiarów pakietu .....	13
c. Analiza .....	13
6. Wnioski .....	14

## *Spis Ilustracji*

<i>Wykres 1 Protokół SAW - kanał BSC - wykres BER i OpTime w zależności od prawdopodobieństwa błędu (dla różnych sposobów kontroli błędów)</i>	<i>7</i>
<i>Wykres 2 Protokół SAW - kanał BEC - wykres OpTime w zależności od prawdopodobieństwa błędu (dla różnych sposobów kontroli błędów)</i>	<i>8</i>
<i>Wykres 3 Protokół SAW - kanał CEC - wykres OpTime w zależności od prawdopodobieństwa błędu (dla różnych sposobów kontroli błędów)</i>	<i>9</i>
<i>Wykres 4 Protokół GBN - kanał BSC - wykres BER i OpTime w zależności od prawdopodobieństwa błędu (dla różnych sposobów kontroli błędów)</i>	<i>10</i>
<i>Wykres 5 Protokół GBN - kanał BEC - wykres OpTime w zależności od prawdopodobieństwa błędu</i>	<i>11</i>
<i>Wykres 6 Protokół GBN - kanał CEC - wykres OpTime w zależności od prawdopodobieństwa błędu (dla różnych sposobów kontroli błędów)</i>	<i>12</i>
<i>Wykres 7 Wykres BER i OpTime w zależności od prawdopodobieństwa błędu dla różnych wielkości pakietu - 8/16 bitów (Protokół SAW - kanał BSC)</i>	<i>13</i>

# 1.Temat

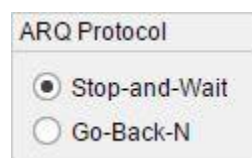
Tematem projektu była symulacja działania i skuteczności systemu kontroli błędów ARQ (ang. Automatic Repeat reQuest). Celem projektu było dowiedzenie skuteczności systemu na dwóch przykładowych protokołach i przeprowadzenie analizy danych w celu wybrania najlepszego modelu dla systemu ARQ. Symulacja została napisana w środowisku Matlab.

## 2. Instrukcja obsługi



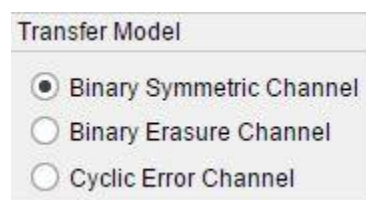
The 'Error Control' panel contains three radio buttons: 'CRC32' (selected), '2z5', and 'Parity Bit'.

Radio box Error Control pozwala wybrać jeden z trzech trybów kontroli błędów. Do wyboru są tryby CRC32, 2z5 oraz Parity Bit (bit parzystości).



The 'ARQ Protocol' panel contains two radio buttons: 'Stop-and-Wait' (selected) and 'Go-Back-N'.

Radio box ARQ Protocol pozwala wybrać jeden z protokołów ARQ. Zaimplementowane zostały Stop-and-Wait oraz Go-Back-N.



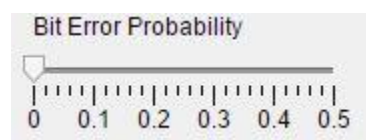
The 'Transfer Model' panel contains three radio buttons: 'Binary Symmetric Channel' (selected), 'Binary Erasure Channel', and 'Cyclic Error Channel'.

Pole Transfer Model pozwala ustalić jaki model zostanie zasymulowany podczas przesyłania danych. W symulacji dostępne są Binary Symmetric Channel (pol. Symetryczny kanał binarny), Binary Erasure Channel (pol. Kanał usuwania bitów), oraz Cyclic Error Channel (pol. Kanał błędów cyklicznego).



Two text input fields: 'Packet Size' with the value '32' and 'Packets Count' with the value '10'.

Za pomocą pól tekstowych Packet Size oraz Packet Count można zdefiniować kolejno rozmiar przesyłanych pakietów, oraz ich ilość.



A slider control for 'Bit Error Probability' ranging from 0 to 0.5. The slider is currently positioned at 0.

Suwak Bit Error Probability służy do płynnego regulowania prawdopodobieństwa wystąpienia błędu w wybranych kanale podczas przesyłania danych. Aby uzyskać znaczące wyniki, należy dobrać wartość

do wybranego kanału.

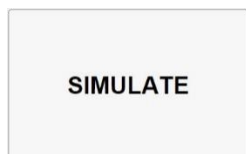


A text input field for 'Transfer rate [b/s]' with the value '1000'.

W polu tekstowym Transfer Rate ustalana jest prędkość transmisji bitów. Wartość ta jest podawana w bitach na sekundę.

Data Save Filename

W polu tekstowym Data Save Filename można podać nazwę docelowego pliku, do którego zostaną zapisane dane. Domyślną nazwą jest „data.txt”. Plik zostaje zapisany do folderu roboczego. Należy podać nazwę pliku, inaczej program nie zadziała.



Przycisk Simulate włącza symulację

Simulation Results

```

BSC;PB;SAW;32;10;1000;0.002;0.000;1.056;2
BSC;PB;GBN;32;10;1000;0.002;0.000;1.155;2
BSC;PB;GBN;64;10;1000;0.002;0.000;1.365;1
BSC;PB;GBN;64;10;1000;0.037;0.034;6.955;7
BSC;PB;GBN;64;10;2000;0.037;0.031;5.363;12
  
```

W polu Simulation Results wyświetlone zostaną wyniki symulacji, które zostaną również zapisane do pliku, którego nazwa jest podana w polu Data Save Filename

### 3. Opis implementacji kodu

#### a. Error Control

##### CRC32 (Cyclic Redundancy Check 32 bit)

W tej symulacji CRC32 (*pol.* cykliczny kod nadmiarowy) wykorzystuje wielomian  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ . Funkcja CRC przyjmuje dane, następnie uzupełnia je zerami do odpowiedniej długości. Następnie skanuje otrzymane dane i jeśli znajdzie 1, wykonuje operację xor na bitach między danymi, a wielomianem. Po przeskanowaniu wszystkich danych, wynik zamieniany jest z wektora na macierz. Funkcja kodująca zwraca wektor zakodowanych danych.

Dekodowanie po otrzymaniu danych działa w analogiczny sposób. Po odkodowaniu operacją xor następuje sprawdzenie warunku, czy ostatnie 32 bity odkodowanego wektora są równe 32 bitom wektora zakodowanego. Jeśli warunek jest spełniony, funkcja zwraca informację o poprawnym przesłaniu danych oraz odkodowane dane w postaci wektora.

Działanie tego systemu opiera się na zasadzie działania operacji xor. Jeśli zostanie zmieniony jeden z bitów w wiadomości, to suma sprawdzająca po odkodowaniu nie będzie równa zakodowanej (poza nielicznymi przypadkami niewykrycia błędu).

##### 2z5

Kodowanie 2z5 polega na uzupełnieniu wysyłanych danych dodatkowymi bitami. Dla 0 są to 11000, zaś dla 1 – 10100. Założenie tego kodu opiera się na prostej logice, iż istnieje mniejsze prawdopodobieństwo przekłamania 5 bitów, niż jednego.

Dekodowanie polega na pobieraniu po 5 bitów z otrzymanej wiadomości. Następuje porównanie znalezionej sekwencji z ustalonymi sekwencjami do kodowania zera i jedynki, oraz zapis odpowiedniej wartości do wektora uncodedData. Jeśli porównanie zwróci fałsz dla obu ciągów, wiemy że bit ten został przekłamany.

### Parity bit (bit parzystości)

Kontrola błędów za pomocą bitu parzystości jest bardzo prostą kontrolą (i często zawodną). Na koniec wiadomości zostaje dodany bit parzystości, obliczany jako suma wszystkich „jedynek” w wiadomości modulo 2. Dekodowanie polega na odcięciu ostatniego bitu z wiadomości, następnie przeliczeniu bitu parzystości dla otrzymanej w ten sposób wiadomości i porównanie wyników. Metoda ta jest zawodna, gdyż jeśli zamienione zostaną dwa (lub dowolna inna parzysta ilość) bity, to test nie wykryje przekłamania.

### *b. ARQ Protocols*

#### Stop-and-Wait

Pierwszy z zaimplementowanych protokołów ARQ, Stop-and-Wait (*pol.* zatrzymaj i poczekaj) wysyła tylko jedną ramkę na raz. Kolejna zostanie wysłana dopiero po otrzymaniu wiadomości ACK (acknowledgement, *pol.* potwierdzenie) od odbiorcy. Jeśli wiadomość nie przyjdzie w zdefiniowanym czasie (w symulacji – dziesięciokrotność matematycznie wyliczonego czasu potrzebnego na przesłanie symulowanej ilości danych z zadaną prędkością), to ramka jest wysyłana ponownie. Jest to bardzo wolny protokół, gdyż przez większość czasu nadawca oczekuje na potwierdzenie odbioru.

Program sprawdza tagi przed każdą ramką informującą o kodowaniach i modelach transferu w jakich wiadomość ma być wysłana, wywołuje odpowiednie metody i przesyła wiadomość. Jeśli nie otrzyma wiadomości ACK, a zgłoszony zostanie timeout, czas operacji zostaje zwiększony, oraz odnotowany zostaje fakt niepoprawnego wysłania ramki

#### Go-Back-N

Protokół Go-Back-N (*pol.* cofnij o N) rozwiązuje największy problem protokołu Stop-and-Wait, czyli jego powolność poprzez wysyłanie kilku ramek naraz. Na początku ustalana jest wielkość okna, w którym przesyłane będą ramki. Następnie, po wywołaniu odpowiednich funkcji kodujących na wiadomości, wiadomość zostaje dołożona do okna i wysłana. W przypadku, gdy któreś z potwierdzeń od odbiorcy nie dojdzie, wysyłane jest na nowo całe okno. Wyjątkiem jest wysyłanie przez kanał BEC, w którym dosyłaniu jedynie brakujących bitów. Czas timeoutu jest obliczany tak samo jak przy protokole Stop-and-Wait.

### *c. Transfer Models*

#### Binary Symmetric Channel (BSC)

Kanał ten może transmitować tylko dwie wartości: 0 lub 1 (stąd nazwa binarny i dlatego alfabetu wejściowy i wyjściowy są identyczne, postaci  $\{0,1\}$ ). Charakteryzuje się zmianą bitu na przeciwny z prawdopodobieństwem  $p$ .

W symulatorze kanał został zaimplementowany z wykorzystaniem funkcji BSC środowiska Matlab.

#### Binary Erasure Channel (BEC)

Kanał binarny erasure (*pol.* usuwanie bitów) usuwa jeden z bitów. Funkcja przyjmuje jako argumenty dane, prawdopodobieństwo usunięcia bitu i aktualny stan wektora. W praktyce usuwanie bitów jest zrealizowane przez wstawianie zdefiniowanego symbolu (w symulacji – „2”) w miejsce straconego bitu. Jeśli aktualny stan wektora wskazuje na utratę niektórych bitów, to bity te są jeszcze raz przesyłane, a aktualny stan aktualizowany. Czynności te są powtarzane aż do momentu, gdy aktualny stan wektora wskaże na brak błędów. Licznik resend zwiększa się, jeśli którykolwiek z bitów w pakiecie jest przesłany.

### Cyclic Error Channel (CEC)

Cyclic Error Channel (*pol.* Kanał błędu cyklicznego) to kanał testowy. Takie zakłócenia rzadko występują naturalnie, jednak był on przydatny przy tworzeniu analiz. Funkcja przyjmuje trzy argumenty, tj. dane, okres błędu i pierwszy bit do przekłamania, i zamienia cyklicznie co  $n$ -ty bit, zgodnie z podanymi argumentami.

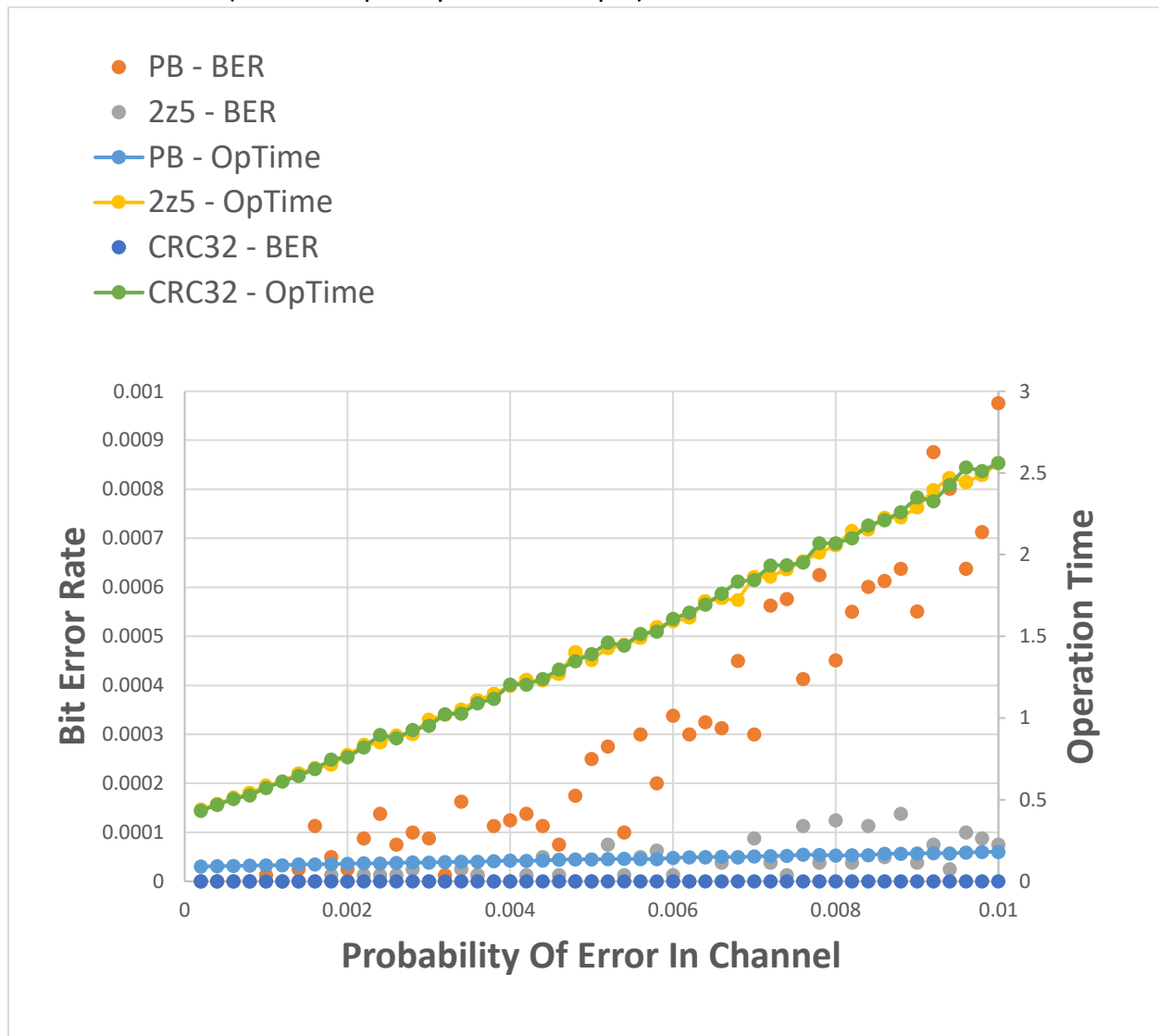
## 4. Opis skryptu uśredniający wynik z wielu prób

W celu przeprowadzenia analizy danych, poza samym symulatorem sporządzono skrypt wykonujący wielokrotne symulacje dla tych samych warunków, w celu uśrednienia wyników empirycznych. W skrypcie można ustalić konfigurację symulacyjną, ilość powtórzeń symulacji oraz zakres prawdopodobieństwa błędu. Dla naszych celów skrypt został zaprogramowany tak, albo wykonywał po 1000 pomiarów dla każdej konfiguracji i zapisywał uśrednione wyniki do pliku. Dla każdego wywołania skryptu wykonuje się 50 testów po 1000 symulacji, w których zwiększa się stopniowo prawdopodobieństwo błędu od  $p=0,0002$  do  $p=0,01$ . Wyjątkiem są testy dla konfiguracji z kanałem CEC (błędów cyklicznych), gdzie przez charakterystykę kanału symulacje mogły być przeprowadzone tylko dla pewnych prawdopodobieństw - od  $p=0,002$  do  $p=0,1$ . W konfiguracji symulacji są także ustawione pewne parametry, których nie zmieniano niezależnie od konfiguracji – Packet Size = 8 oraz Packets Count = 10. Takie ustawienie jest głównym powodem, dla którego kanał transmisji z błędami cyklicznymi mógł działać tylko dla określonych wartości błędu - jeżeli błąd był za niski, to błąd występował co okres większy niż  $8 \cdot 10 = 80$  bitów, jeżeli za duży, to błąd występował częściej, niż rozmiar przesyłanego, zakodowanego pakietu - więc nie było szansy na przesłanie prawidłowej całości.

## 5. Analiza Wyników:

### a. Protokół SAW

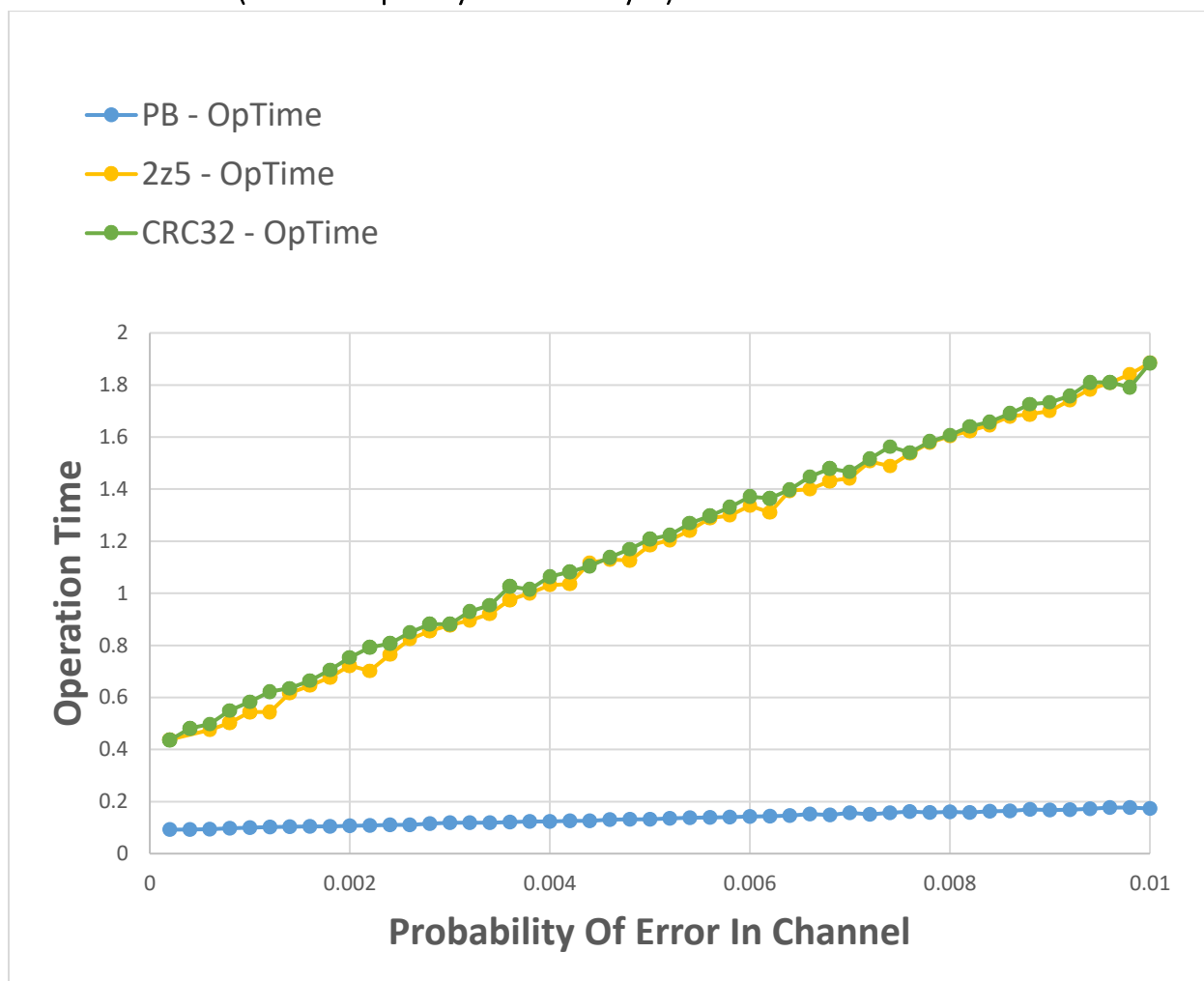
Kanał BSC (na 1 osi 3 sposoby kontroli danych)



Wykres 1 Protokół SAW - kanał BSC - wykres BER i OpTime w zależności od prawdopodobieństwa błędu (dla różnych sposobów kontroli błędów)

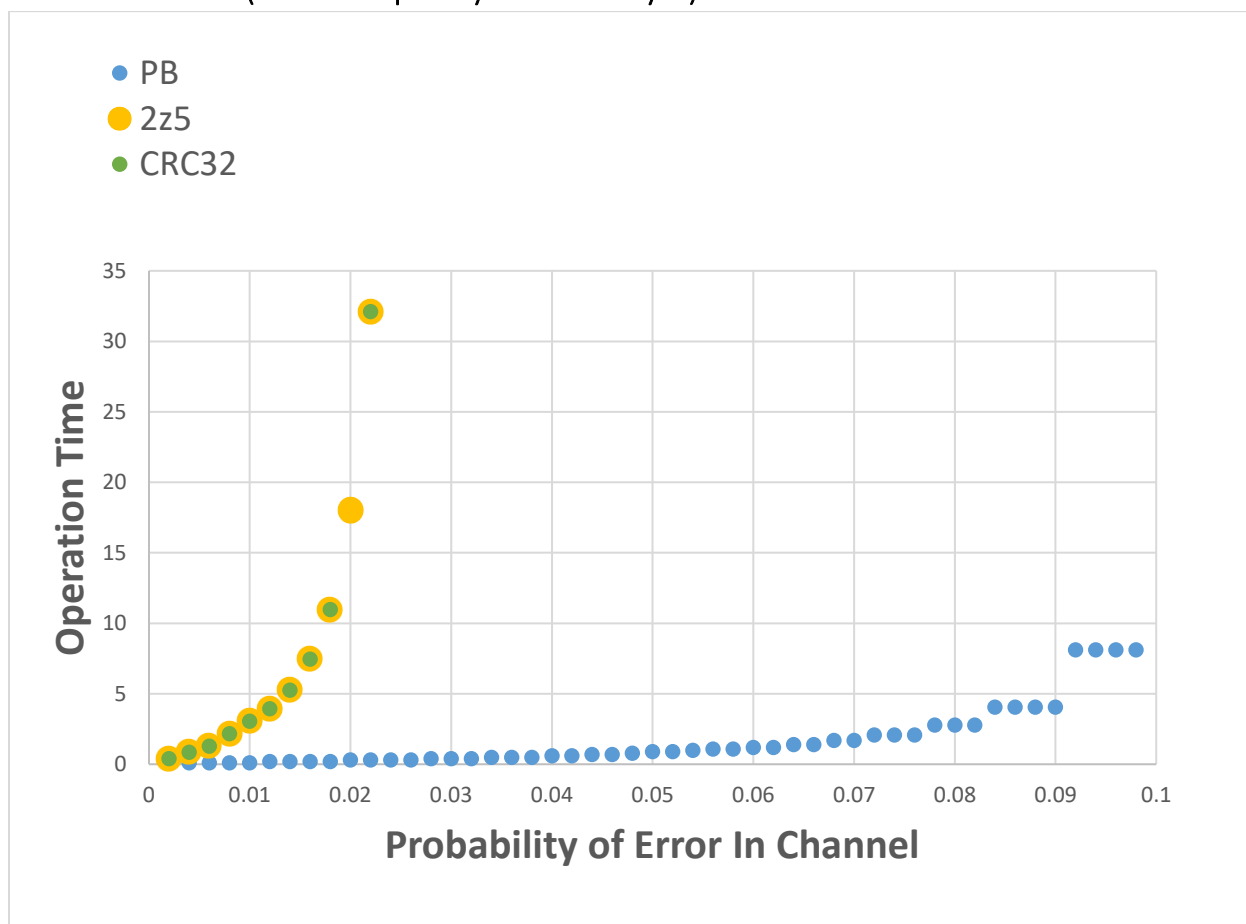


### Kanał BEC (na 1 osi 3 sposoby kontroli danych)



Wykres 2 Protokół SAW - kanał BEC - wykres OpTime w zależności od prawdopodobieństwa błędu (dla różnych sposobów kontroli błędów)

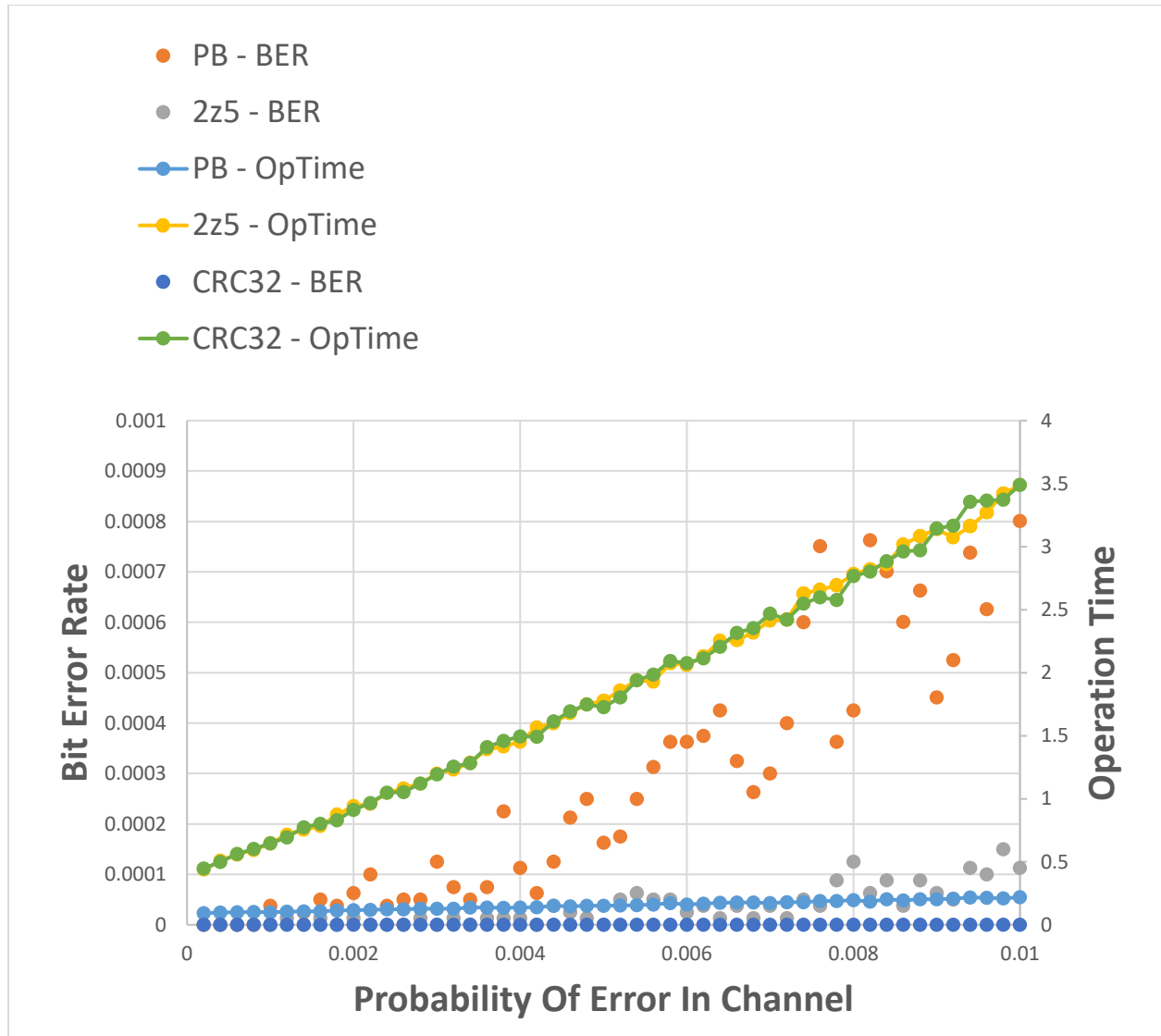
### Kanał CEC (na 1 osi 3 sposoby kontroli danych)



Wykres 3 Protokół SAW - kanał CEC - wykres OpTime w zależności od prawdopodobieństwa błędu (dla różnych sposobów kontroli błędów)

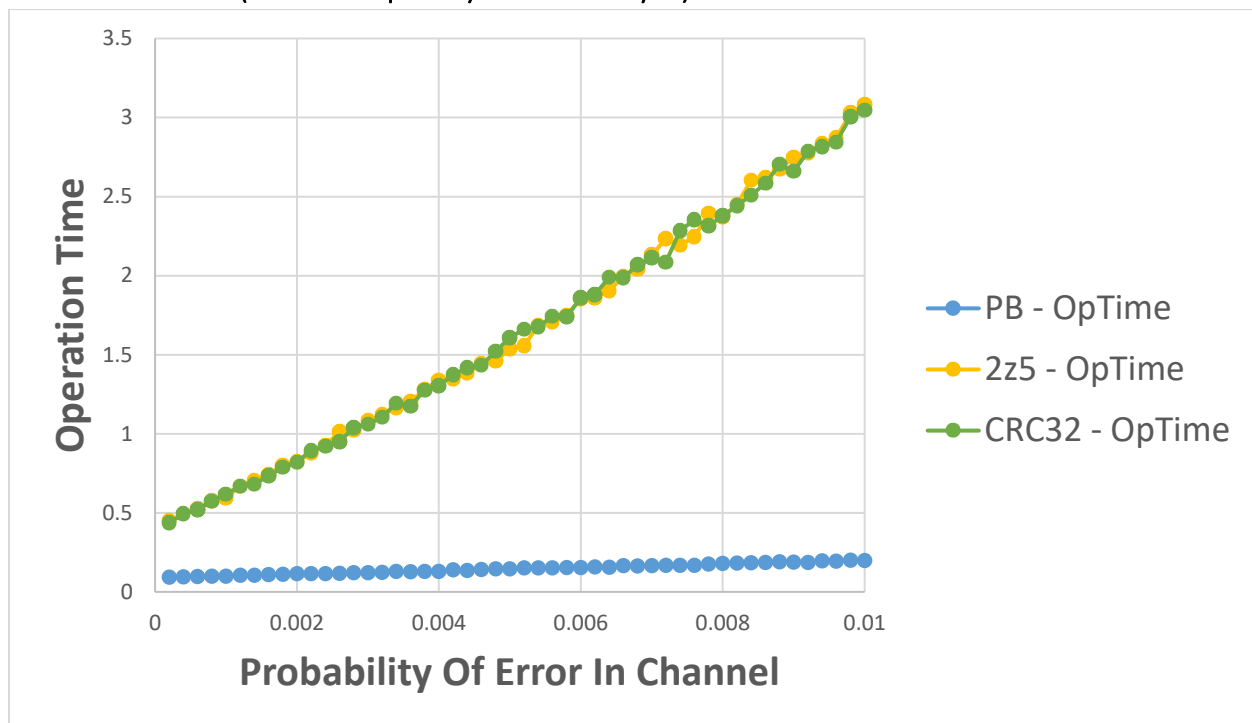
b. Protokół GBN

Kanał BSC (na 1 osi 3 sposoby kontroli danych)



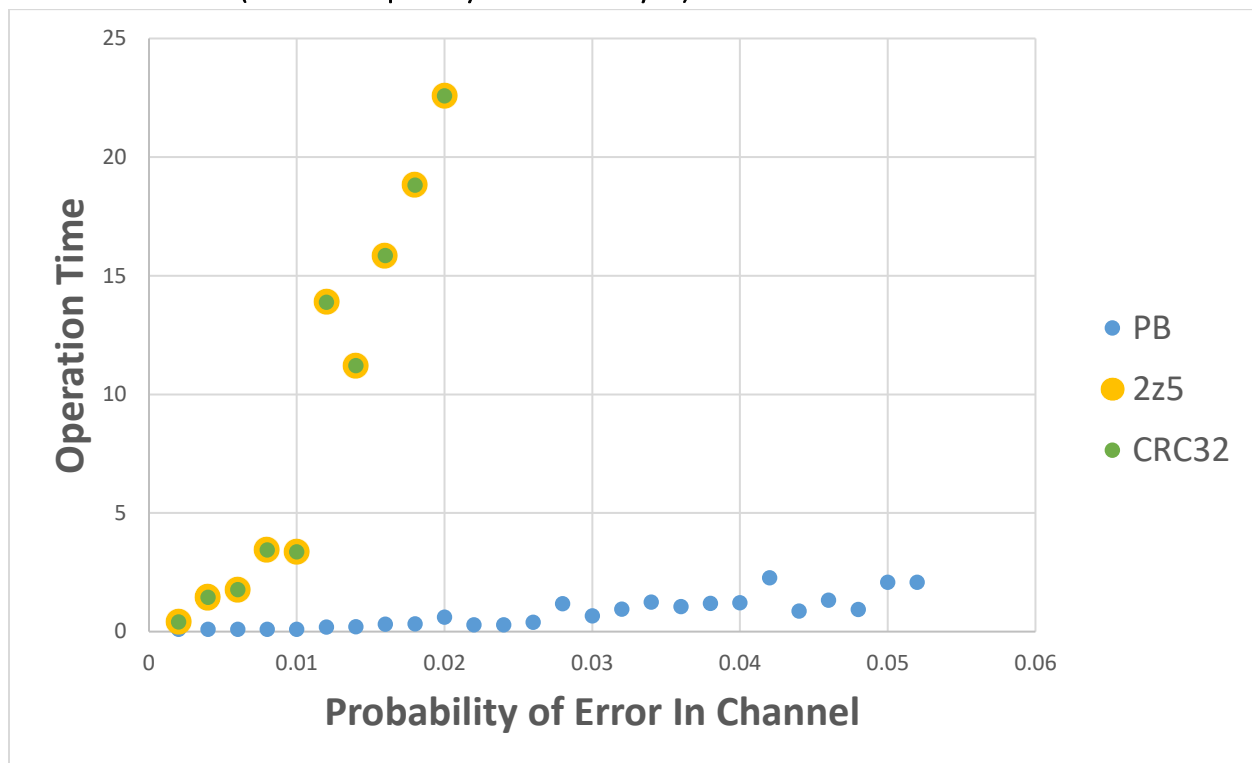
Wykres 4 Protokół GBN - kanał BSC - wykres BER i OpTime w zależności od prawdopodobieństwa błędu (dla różnych sposobów kontroli błędów)

### Kanał BEC (na 1 osi 3 sposoby kontroli danych)



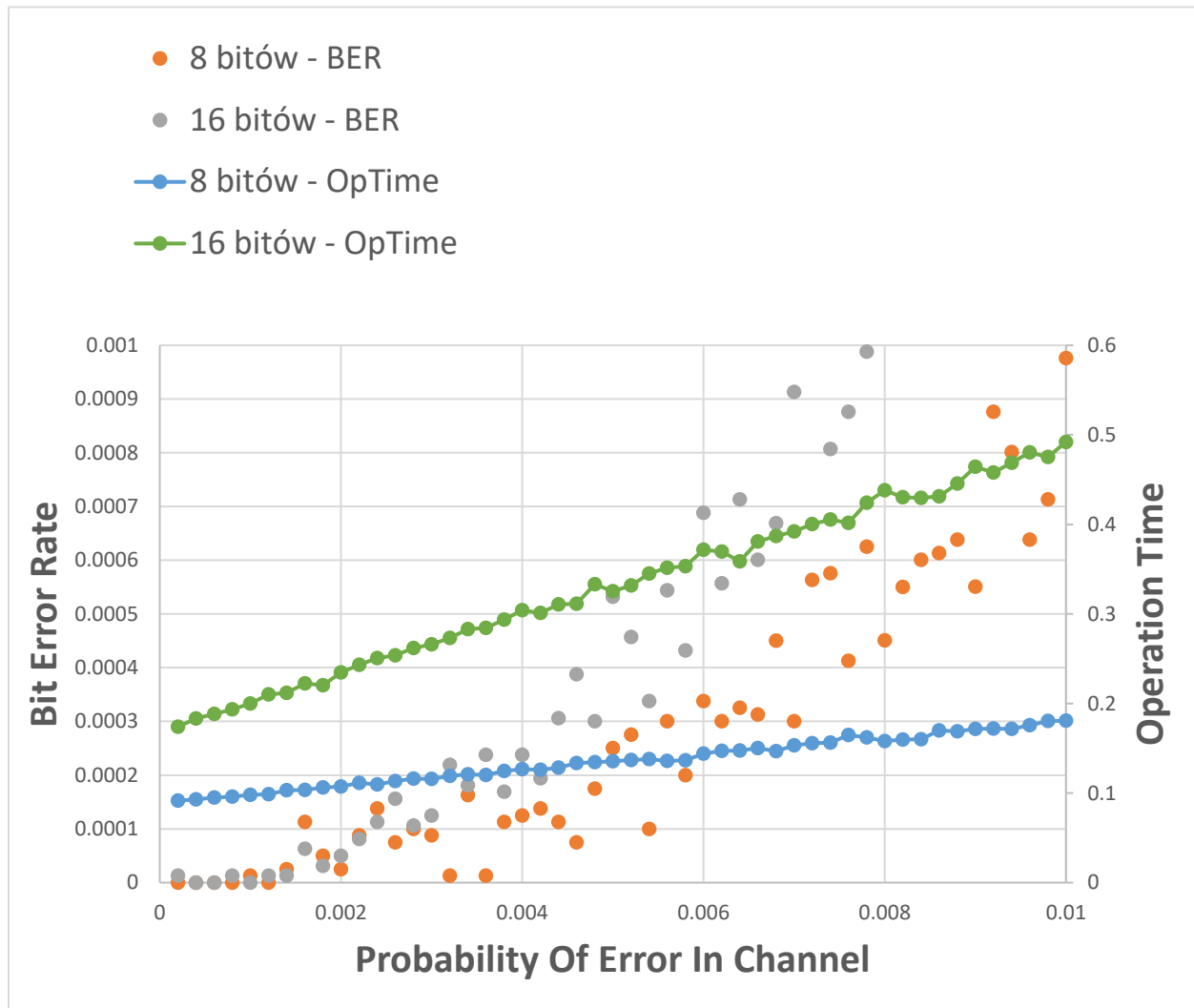
Wykres 5 Protokół GBN - kanał BEC - wykres OpTime w zależności od prawdopodobieństwa błędu

### Kanał CEC (na 1 osi 3 sposoby kontroli danych)



Wykres 6 Protokół GBN - kanał CEC - wykres OpTime w zależności od prawdopodobieństwa błędu (dla różnych sposobów kontroli błędów)

c. Dodatkowo - działanie PB dla 2 różnych rozmiarów pakietu



Wykres 7 Wykres BER i OpTime w zależności od prawdopodobieństwa błędu dla różnych wielkości pakietu - 8/16 bitów (Protokół SAW - kanał BSC)

### c. Analiza

Przed wszystkim, porównując 2 protokoły, SAW i GBN, można zauważyć, że czas przesyłania dla SAW jest krótszy. W przypadku użycia BSC, BER jest podobny (delikatna przewaga dla SAW).

- Dla kanałów BEC i CEC uzyskiwano tylko i wyłącznie BER = 0. Jest to zdecydowanie dobra cecha obu protokołów. Dla błędów cyklicznych, problemem może się okazać zbyt duża częstotliwość błędów, która nie będzie pozwalała na odpowiednie przesłanie żadnego z pakietów - wtedy dobrym rozwiązaniem jest próba zmniejszenia rozmiaru pakietu.
- Dla kanałów BSC i BEC czasy operacji rosną liniowo, z współczynnikiem kierunkowym mniejszym od 1. W BSC jest to (jak wynika z regresji liniowej sporządzonej na podst. wykresu) około 1/10, w BEC nawet mniej - może to wynikać z naszej implementacji kanałów.

- Na przykładzie różnicy implementacji kanałów BSC i BEC widać, że opłaca się wysyłać tylko błędne bity, lub generalnie 'zawęzić' pakiety wysyłane 'ponownie', gdyż zmniejsza to czas transmisji.
- Zgodnie z oczekiwaniami dla większych rozmiarów pakietów, kodowanie z pomocą bitu parzystości generuje większą stopę błędów.

## 6. Wnioski

Przeprowadzone symulacje i analiza pozwoliły wyciągnąć wszelakie wnioski.

- Środowisko Matlab pozwala z wielką łatwością tworzyć tego typu symulacje, zwłaszcza dla problemów wymagających działania na wielu ciągach liczb, które tutaj można z wielką łatwością przedstawić jako macierze i wektory.
- Symulacja wykazała, że protokół Stop-and-Wait działa szybciej niż protokół Go-back-N dla kanałów BSC i BEC. Jest to szokujący wynik, w stosunku do tego, czego można było się spodziewać, jednak może on być związany przede wszystkim z wielkością okna w protokole Go-back-N oraz ustawień timeoutu w protokole Stop-and-Wait i prędkości przesyłu bitów, które użyto podczas testów.
- W przypadku kanału Erasure najważniejszym kryterium przy wybieraniu protokołu jest czas operacji, gdyż bity są wysyłane tak długo, aż pakiet zostanie odebrany poprawnie. Z symulacji wynika, że najlepszym kodowaniem do tego kanału jest najprostsze, czyli bit parzystości z racji najkrótszego czasu operacji.
- Podobna sytuacja (odnośnie czasu operacji) występuje w kanale CEC, w którym czas operacji dla kodowań 2z5 i CRC32 rośnie eksponencjalnie, a dla bitu parzystości dużo wolniej (liniowo, z bardzo małym współczynnikiem).
- Kodowanie bitem parzystości nie nadaje się do kanału BSC, w którym stopa błędów przy tym sposobie rośnie drastycznie w obu protokołach. Tutaj dużo lepiej sprawdzają się sumy kontrolne i metoda 2z5.