

NIEZAWODNOŚĆ I DIAGNOSTYKA SYSTEMÓW CYFROWYCH

Streszczenie

Transmisja w systemie ARQ na podstawie pliku JPG.

Anna Trębicka
Michał Moskała
Paweł Szynal

Zadanie projektowe nr 1

Temat projektu: 1. Transmisja w systemie ARQ (Automatic Repeat Request)			
Data wykonania ćwiczenia	16.06.2017	Termin zajęć	Poniedziałek TN 14 ⁰⁰ – 17 ⁰⁰
Data oddania sprawozdania	16.06.2017	Wersja	1
Skład grupy		Prowadzący	Ocena
Anna Trębicka 226036 Michał Moskała 226107 Paweł Szynal 226026		Dr inż. Jacek Jarnicki	

1 Cel projektu:

Celem projektu jest zaprojektowanie i napisanie oprogramowania umożliwiającego symulację pracy systemu ARQ na podstawie przesyłania plików graficznych o formacie JPG. Wykorzystane wyniki analizy syntetycznej do optymalizacji parametrów systemu.

2 Etapy realizacji projektu:

1. Zbudowanie modelu symulacyjnego i symulatora systemu transmisji Informacji
2. Wykonanie eksperymentu symulacyjnego i rejestracja jego wyników,
3. Statystyczna analiza wyników eksperymentu,
4. Optymalizacja wybranych parametrów systemu
5. Opracowanie sprawozdania końcowego.

3 Założenia:

- Generowane dane będą pochodzić plików graficznych o formacie JPG.
- Wybrany środowiskiem programistycznym jest Eclipse a program napisany jest w całości w języku JAVA.

4 Podział zadań projektowych.

Anna Trębicka – testy , badanie plików JPG oraz dobór parametrów algorytmów ARQ.

Michał Moskała – implementacja algorytmów haszowania oraz transmisji.

Paweł Szynal - implementacja algorytmów dekodowania, retransmisji oraz stworzenie interface'u graficznego programu.

Spis treści

Zadanie projektowe nr 1	1
Wstęp teoretyczny:	3
Automatic Repeat reQuest (ARQ)	3
Uproszczony model projektu	3
Działanie systemu:	4
Wykorzystane algorytmy:	5
Kanał Parity	5
Przykład działania:	6
Kanał Modulo:	6
Kanał Multiply:	6
Ogólnie działanie:	7
Efektywność systemu / BER:	7
Generowanie błędów:	7
Testowanie algorytmów:	8
Kanał Parity	9
Wybrane wyniki testowań:	9
Wnioski:	11
Kanał Modulo	13
Wybrane wyniki testowań:	13
.....	15
Wnioski:	15
Kanał Multiply	17
Wybrane wyniki testowań:	17
Wnioski:	19
Błędy napotkane w trakcie testowania programu:	21
Błąd bitowy w skanerze JPEG (kolor - błąd)	21
Uszkodzony nagłówek JPG	21
Wnioski:	22

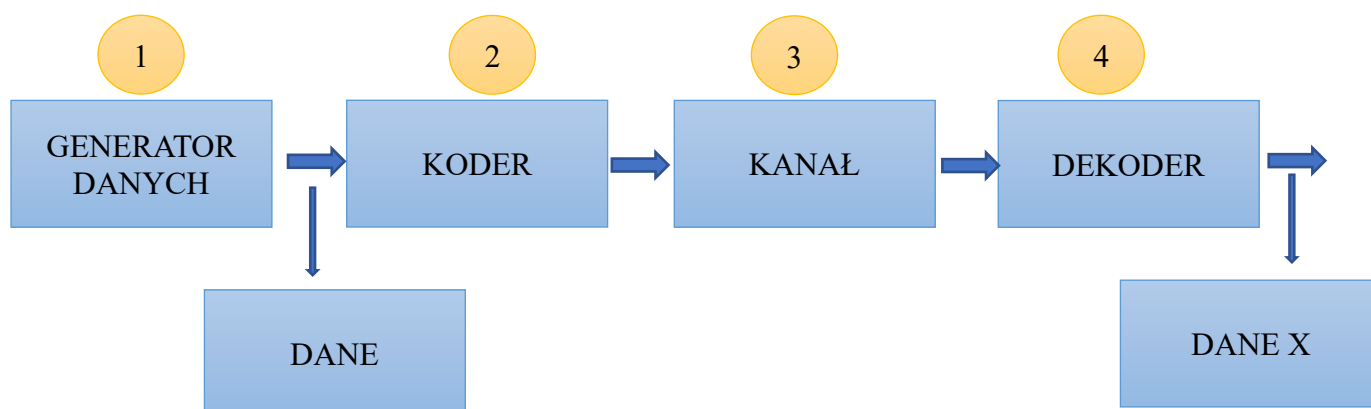
Wstęp teoretyczny:

Aby system został uznany przez użytkownika za udany konieczne jest aby jego dwa parametry: szybkość transmisji i wierność odbieranych danych były jak najlepsze. W celu zapewnienia dobrej jakości przesyłania informacji z pomocą przychodzi kodowanie nadmiarowe czyli dopisywanie do bitów informacyjnych, w pewien kontrolowany sposób (zależny od rodzaju kodu) dodatkowych (nadmiarowych) bitów. W odbiorniku, bity nadmiarowe mogą zostać wykorzystane do detekcji błędów albo do korekcji błędów (system FEC - Forward Error Correction). Ponieważ systemy kodowane korekcyjnie nie gwarantują uzyskania zerowej liczby błędów na wyjściu odbiornika, w systemach przeznaczonych do przesyłania danych stosuje się wykrywanie błędów połączone z retransmisją błędnych bloków czyli system ARQ. Wykrywanie błędów polega na sprawdzeniu, czy została zachowana reguła, według której dopisano w nadajniku nadmiarowe bity (na podstawie odebranych bitów informacyjnych ponownie wylicza się bity nadmiarowe, a następnie porównuje je z odebranymi). Istnieją trzy podstawowe warianty systemu ARQ: stop-and-wait (SAW), go-back-N (GBN) oraz selective repeat (SR).

Automatic Repeat reQuest (ARQ)

System ARQ charakteryzuje parametr zwany efektywnością lub sprawnością systemu. Liczony jest on, jako stosunek liczby odebranych bitów informacyjnych do liczby wszystkich możliwych do przesłania bitów. Efektywność systemu ARQ zależy od jakości kanału, długości pojedynczego bloku oraz od wybranej metody ARQ.

Uproszczony model projektu



Działanie systemu:

Danymi wejściowymi, które w dalszej części projektu będziemy przetwarzać jest obraz formatu JPEG. Początkowo dane przechowywane w tym formacie zostają przepisane na ciąg bitów, który to trafia do kanału. Bity przed wysłaniem zostają podzielone na paczki (dalej w projekcie nazywane ramkami), które program będzie sprawdzał. W opcjach programu możemy wybrać wielkość ramki – 8, 16 oraz 32. Oczywiście wielkość ramki wpłynie na jakość transmisji oraz ilość występujących błędów – co doskonale pokazuje późniejsze testowanie programu. Przed wysłaniem ostatecznych danych do odkodowania i zamienienia powrotnie na plik JPEG, program sprawdza ich poprawność – wzorując się na systemie ARQ.

W napisanym przez nas projekcie będziemy korzystać z lekko zmodyfikowanej metody

SR – Selective Repeat. Polega ona na początkowym przesłaniu przez kanał wszystkich ramek. Po tej czynności program sprawdza, które ramki zostały wysłane poprawnie i prosi o ponowne przesłanie niektórych z nich. Program przesyła powtórnie tylko błędne ramki i znów sprawdza ich poprawność. Ramki zostają w ten sposób wysyłane dopóki jest przekroczony dopuszczalny próg błędnych danych (w programie zmienna *errorallowance* ustawiana jako jeden z parametrów kanału). Na koniec dekodery zamienia otrzymane bity na dane pliku JPEG i generuje obraz po transmisji.

Napisana przez nas aplikacja pozwala na porównanie zdjęcia źródłowego (source) ze zdjęciem, które powstałoby przechodząc przez kanał bez transmisji ARQ oraz z transmisją (Rys 1.). Po prawej stronie zostaje także wyświetlony BER – Bit Error Rate, czyli stosunek procentowy uszkodzonych danych do pozostałych. Wyświetlamy go w dwóch wariantach: dla transmisji ARQ oraz bez niej.



Rys 1. Przykładowy wynik transmisji zdjęcia JPEG

Wykorzystane algorytmy:

- Algorytm „Parity”.
- Algorytm „Modulo”
- Algorytm „Multiply”

Początkowo planowane zostało wykonanie algorytmu Parity, opartego na sprawdzaniu poprawności ramki na podstawie bitu parzystości. Jednak po głębszym przeanalizowaniu sposobu działania powyższego kanału postanowiliśmy napisać jeszcze dwa inne. Pierwszy z nich oparty jest na systemie obliczania modulo 16. Algorytm Multiply, jest autorskim pomysłem naszego zespołu. Wszystkie trzy algorytmy zostaną opisane poniżej.

Prowadzący zalecił również zapoznać się z algorytmem skrótu SHA2 i spróbować go wykorzystać. Jednakże pomimo idei SHA2, która służy do generowania krótkiego zestawu kryptograficznego dla rozległej reprezentacji bitów, wykorzystanie SHA2 jest niepraktycznym zastosowaniem dla małych ramek. Dowodem na to jest wydłużenie pracy algorytmu ARQ, ponieważ dla każdej ramki SHA2 generuje tablicę 256 bitów. W naszym projekcie wykorzystujemy niewielkiej długości ramki :

- 8 ;
- 16 ;
- 32 ;

Kanał Parity

Jest to najprostszy, a co za tym idzie najmniej dokładny, algorytm wykrywania błędnych danych. Polega na dodaniu do każdego poszatkowanego ciągu bitów tzw. bitu parzystości (o wartości 1) a następnie sprawdzeniu (czyli zsumowaniu wszystkich „jedynek”) parzystości danego ciągu. Jeśli w danych wejściowych parzystość jest taka sama jak po przejściu danych przez kanał, informacje uważane są za przesłane poprawnie. Nietrudno zauważyć na podstawie tego uproszczonego opisu, że bit parzystości nie jest dokładnym sprawdzeniem poprawności przesyłanych danych, ponieważ gdy w jednej ramce zostaną przekłamanie dwa bity (bądź ich parzysta ilość) suma przed i po przesłaniu ma tą samą parzystość i błąd nie zostanie wykryty.

Przykład działania:

Dane wejściowe:

1100101001101011 | 1 <- dodanie bitu parzystości

Dane wyjściowe (bez przekłamania)

1100101001101011 | 1

Suma wejściowa: parzysta, Suma wyjściowa: parzysta
Błędu nie wykryto (poprawnie)

Dane wyjściowe (przekłamanie 1 bit)

1000101001101011 | 1

Suma wejściowa: parzysta, Suma wyjściowa: nieparzysta
Wykryto błąd (poprawnie)

Dane wyjściowe (przekłamanie 2 bity)

1010101001101011 | 1

Suma wejściowa: parzysta, Suma wyjściowa: parzysta
Algorytm nie wykryje błędu (niepoprawnie)

Kanał Modulo:

```
static short modulus(String s) {  
    int a = Integer.parseInt(s, 2);  
    a = a % 16;  
    short b = (short) a;  
    return b;  
}
```

Kanał Multiply:

```
static short multipli(String s) {  
    int a = Integer.parseInt(s, 2);  
    double d = Math.PI;
```

```

d = d * a;
long l = (long) d;
double fract = d - l;
d = fract * 16;
short b = (short) d;

return b;
}

```

Ogólnie działanie:

- Generacja danych;
- Wygenerowane dane trafiają do nadajnika;
- Odbieranie danych przez odbiornik;
- Sprawdzanie poprawności otrzymanych danych (Parity, Modulo, Multiply).

Efektywność systemu / BER:

Z punktu widzenia użytkownika, jakość systemu ARQ charakteryzują dwa parametry: efektywność systemu oraz stopa błędów (w programie jako BER – Bit Error Rate). Efektywność systemu liczony jest jako stosunek liczby odebranych bitów informacyjnych do liczby wszystkich możliwych do przesłania bitów, czyli jako stosunek szybkości efektywnej (szybkości transmisji z punktu widzenia użytkownika końcowego) do fizycznej szybkości w kanale. Przykładowo, efektywność wynosząca 0.5 oznacza, że jeżeli w kanale przesyłamy dane z szybkością 64 kbit/s, dla użytkownika system widziany jest jakby pracował jedynie z połową szybkości, czyli 32 kbit/s. Efektywność systemu ARQ zależy od jakości kanału, długości pojedynczego bloku oraz od wybranej metody ARQ.

```

static double calculateBitErrorRate(String imageInString, String imageFromString) {
    int ber = 0;
    char c, c2;
    for (int n = 0; n < imageInString.length(); n++) {
        System.out.println(n + "/" + (imageInString.length() - 1));
        c = imageInString.charAt(n);
        c2 = imageFromString.charAt(n);
        if (c != c2)
            ber++;
    }
    double b1 = ber;
    b1 = b1 / imageInString.length();
    b1 = b1 * 100;
    return b1;
}

```

// obliczana wartość wyrażona jest w procentach

Generowanie błędów:

W celu wygenerowania przekłamań w kanale transmisyjnym wykorzystujemy rozkład Gaussa pobieranego z sekwencji generatora liczb losowych. Powyższa metoda wykorzystuje metodę polarną jak opisał Donald E. Knuth w sztuce programowania komputerowego.

Wygenerowaną pseudolosową liczbę dzielimy przez parametr odpowiadający jakości kanału aby następnie dodać ją do woltażu.

Przekroczenie połowy jej wartości odpowiada bitowi '1', zaś w odwrotnej sytuacji przyjmuje wartość równą '0'.

```
for (int n = 1; n < imageInString.length(); n++) {
    System.out.println(n + "/" + imageInString.length());
    c = imageInString.charAt(n);
    val = Character.getNumericValue(c);
    val = val * 5;

    Random random = new Random();
    d = random.nextGaussian();
    d = d / jakosc_kanal;
    val = val + d;

    if (val > 2.5)
        c = '1';
    if (val <= 2.5)
        c = '0';

    s = Character.toString(c);
    imageFromString = imageFromString.concat(s);
}
```

Testowanie algorytmów:

W celu sprawdzenia jakości kanałów i próby optymalizacji stworzonej transmisji ARQ wykonane zostało testowanie programu. Przetestowano 3 stworzone kanały: Parity, Modulo oraz Multiply w trzech jakościach: dobrej, średniej i złej. Sprawdzano także jak wielkość ramek wpływa na jakość transmisji oraz jak działają poszczególne kanały przy różnych dopuszczalnościach ostatecznych błędów. Sprawdzono przy jakich wartościach powyższych parametrów obrazu nie da się odtworzyć po przejściu przez kanał.

Testowanie zostało wykonane w pętli powtarzającej się 100 razy.

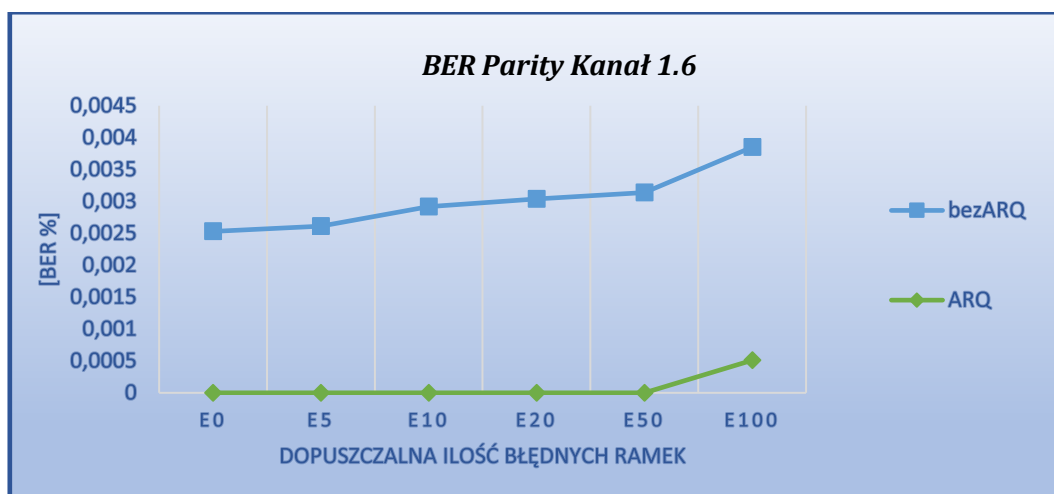
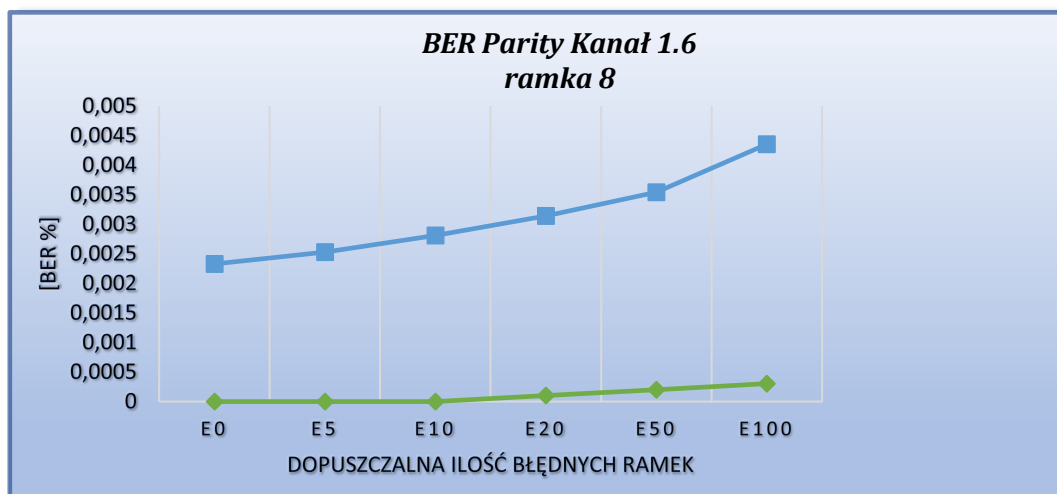
Wyniki uśredniono a wykresy zostały zrobione przy pomocy programu Excel.

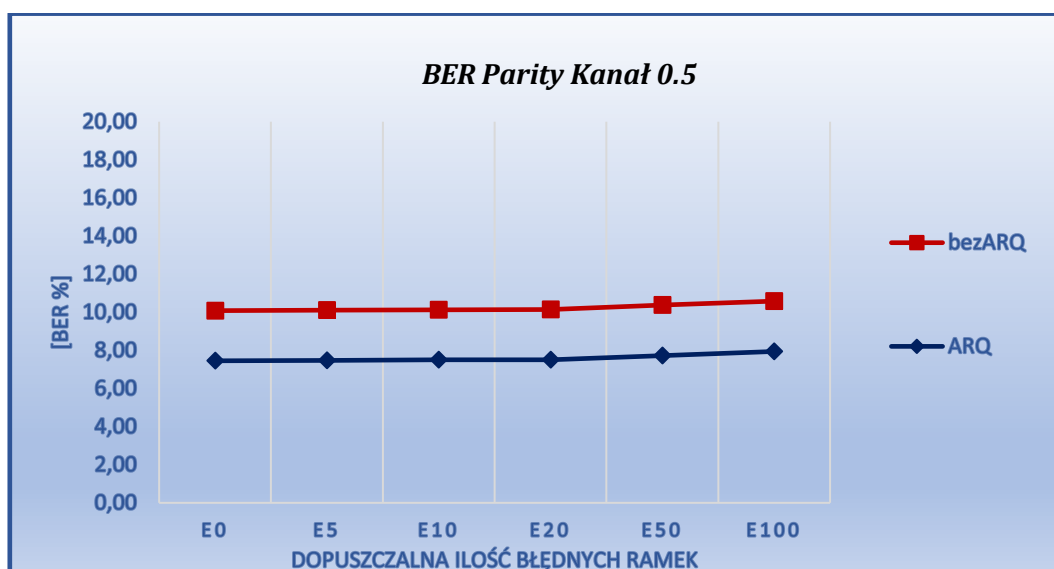
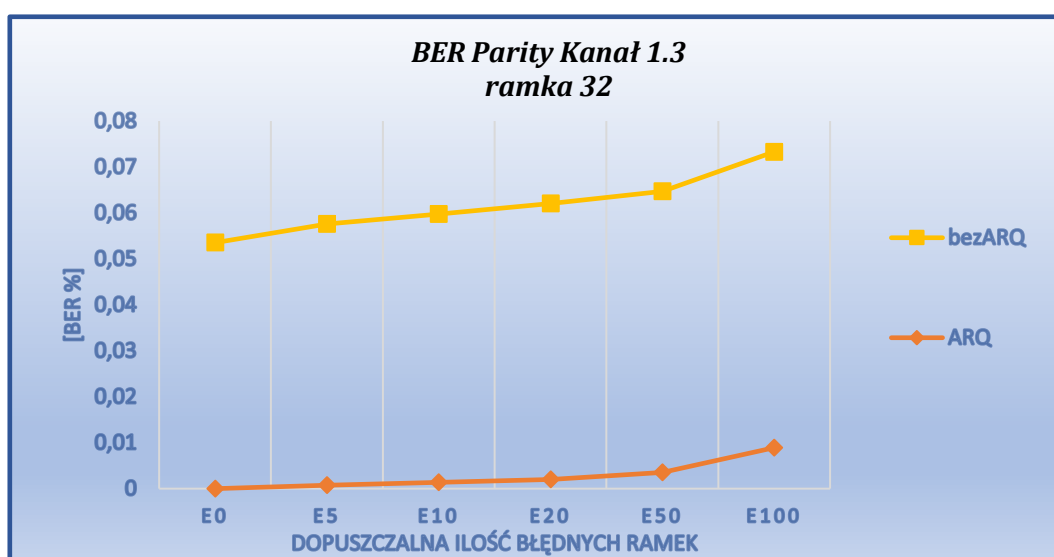
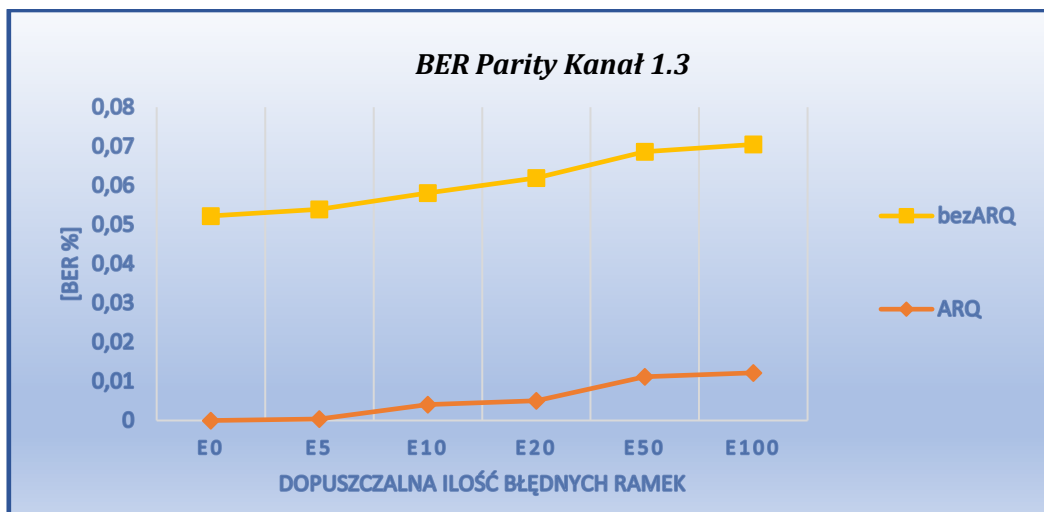
W trakcie testowania napotkaliśmy też wiele błędów związanych z obsługiwany typem pliku: JPEG. Dane są tu kodowane w specyficzny sposób. Dla przykładu: główne informacje o pliku zawierają się w początkowych bitach – nagłówku. Gdy nastąpi przekłamanie w tym miejscu, nawet dla poprawnego kanału, obraz nie odtworzy się, ponieważ najważniejsze dane zostały zniekształcone. Problemy przy testowaniu dokładniej przedstawione zostaną w dalszej części sprawozdania.

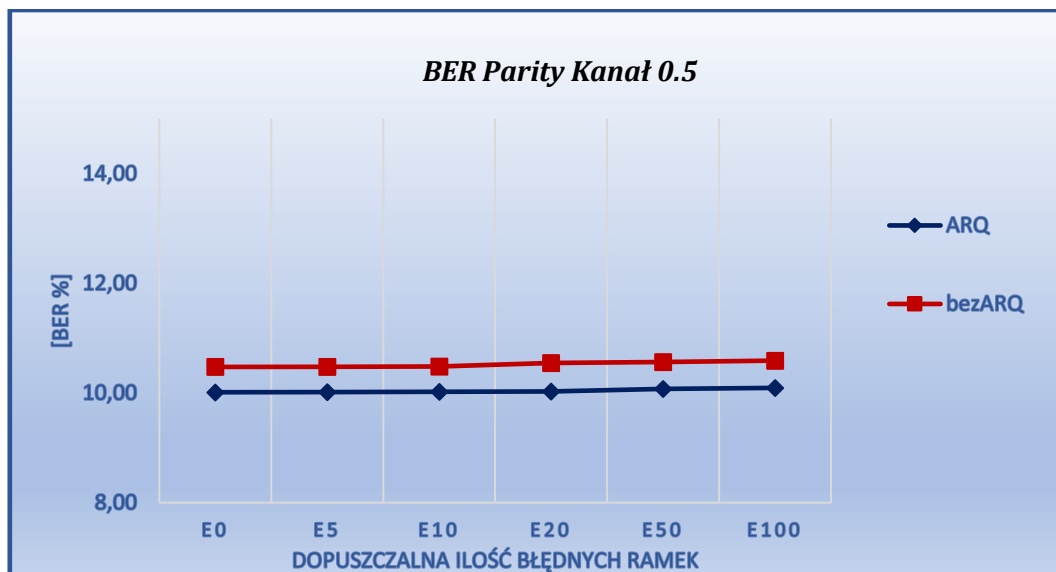
Kanał Parity

Wybrane wyniki testowań:

Testowanie wielkości BER dla zwiększającej się ilości dopuszczalnych błędów







Wnioski:

Na powyższych wykresach bardzo dobrze widać, że algorytm bazujący na sposobie bitu parzystości nie jest dokładny i przepuszcza dużo błędnych ramek, traktując je jako poprawne. Możemy też zauważyć, że w przypadku bardzo złego kanału, algorytm zupełnie nie spełnia swojego zadania, ponieważ transmisja po ARQ jest obciążona tylko lekko mniejszym BER niż bez niej, a i tak oscyluje w okolicach 10%. W przypadku złego kanału zdjęcia nie udało się ponownie odtworzyć.

Można też zauważyć, że wielkość ramki ma wpływ na dokładność i poprawność przesyłanych danych. Przy dużych ramkach większa jest szansa na wystąpienie parzystej ilości przekłamanych bitów, przez co obraz odtwarza się w znacznie gorszej jakości nawet po transmisji.

Przy dobrej jakości kanału zdjęcia po transmisji otwierały się nawet przy dopuszczalnej ilości błędów w okolicach 50-100 (choć czasem z jakimś dobrze widocznym błędem np. zaciemnieniem połowy obrazu). Przy średniej i złej jakości kanału, program miał problem z odtworzeniem zdjęcia już przy 20 dopuszczalnych błędach. Ponadto przy złym kanale zdjęcia nie dało się wygenerować w ogóle nawet po ARQ.

Metoda bitu parzystości jest więc dobra tylko dla kanałów o dobrej jakości z jak najbardziej poszatkowanymi danymi (więcej ramek o mniejszej długości).

Przykładowe rezultaty:

ARQ

Wybór Kanału

Parity

Jakość kanału

Dobry

Ilość dopuszczalnych błędów

5




Ramka

8

Source image

Zdjęcie bez transmisji ARQ

Zdjęcie z transmisją ARQ



START

BEZ ARQ 0.0030373902742763417%

ARQ 0.0030373902742763417%

Rys. 2 Nawet przy dobrej jakości kanału zdarzały się dość poważne błędy w transmisji.

ARQ

Wybór Kanału

Parity

Jakość kanału

Średni

Ilość dopuszczalnych błędów

5


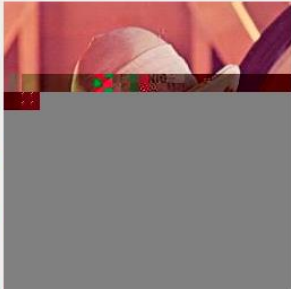
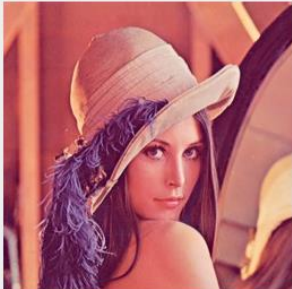
Ramka

8

Source image

Zdjęcie bez transmisji ARQ

Zdjęcie z transmisją ARQ



START

BEZ ARQ 0.05669795178649171%

ARQ 0.0%

Rys.3 Przy średniej jakości kanału, małej ramce i małej ilości dopuszczonych błędów udawało się, że program zadziałał poprawnie, jednak zaobserwowano tu duży element losowości.

ARQ

Wybór Kanału

Parity

Jakość kanału

Zły

Ilość dopuszczalnych błędów

5

Ramka

8

Source image

Zdjęcie bez transmisji ARQ

Zdjęcie z transmisją ARQ

START

BEZ ARQ 10.497220787899037%

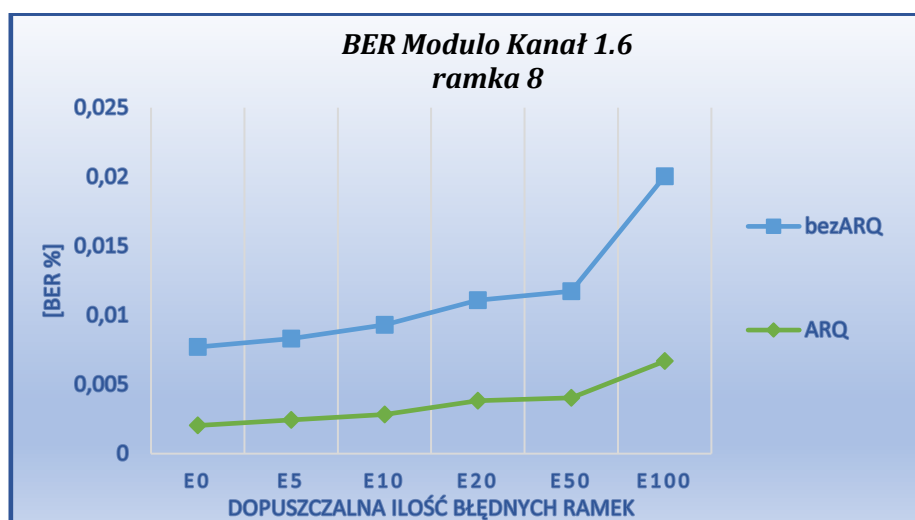
ARQ 7.234051169901487%

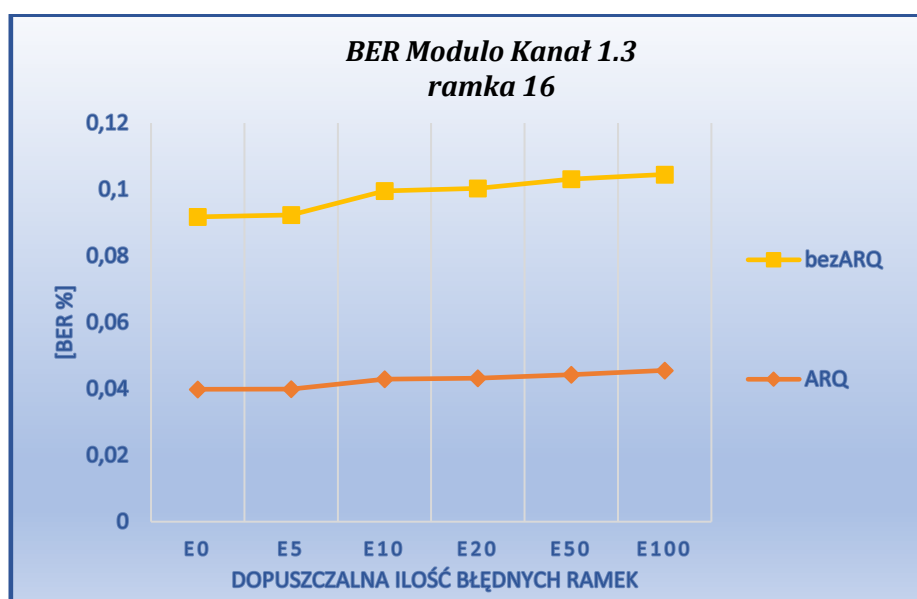
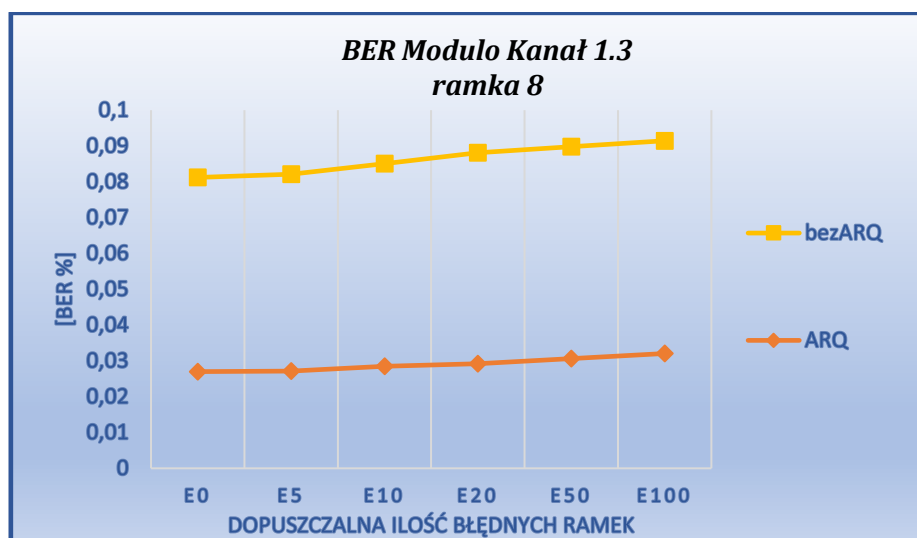
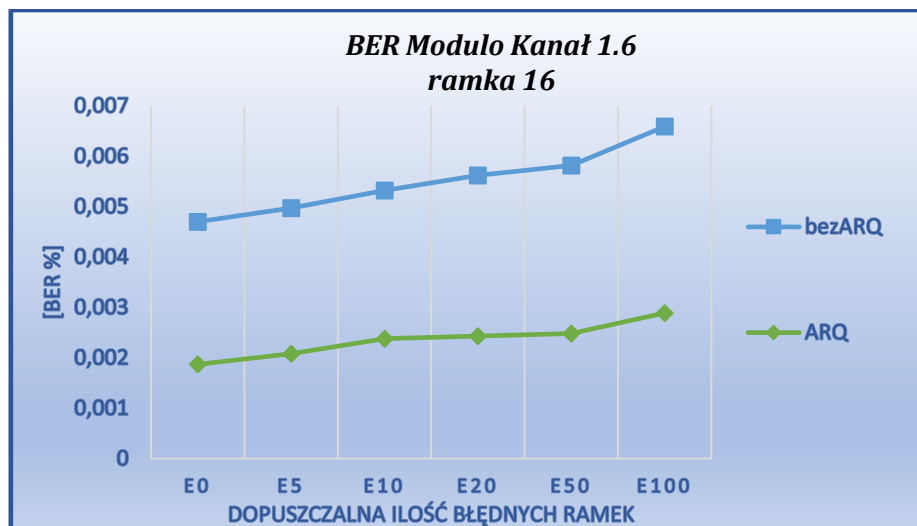
Rys. 4 Przy złym kanale, nawet dla małej ilości dopuszczonych błędów ARQ zmniejszało ilość błędnych ramek o maksymalnie 3 punkty procentowe, co nie pozwalało na odtworzenie i poprawne przesłanie pliku.

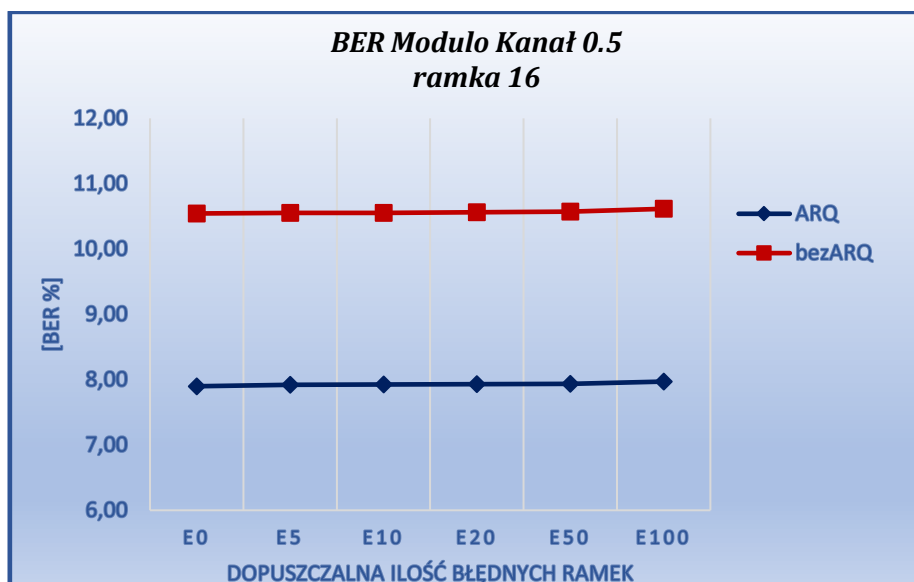
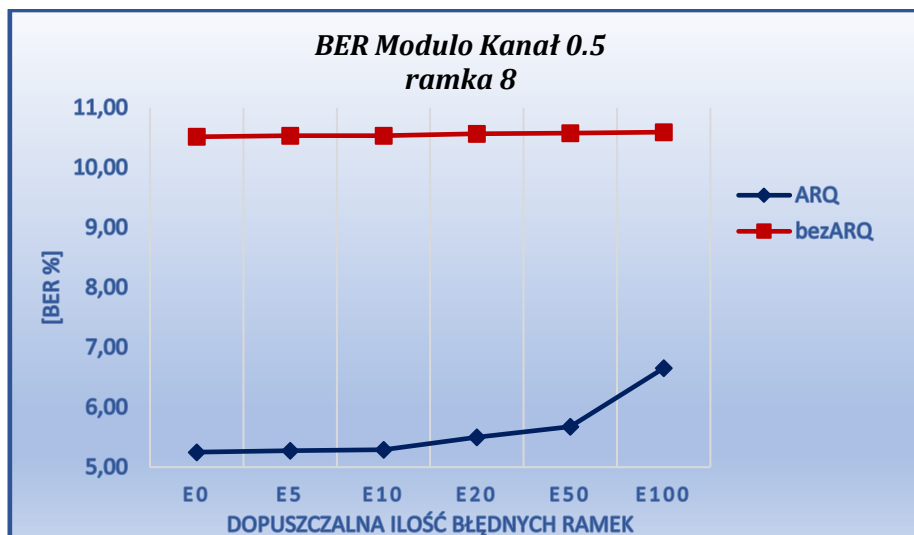
Kanał Modulo

Wybrane wyniki testowań:

Testowanie wielkości BER dla zwiększającej się ilości dopuszczalnych błędów







Wnioski:

Po powyższych wynikach widać, że kanał modulo lepiej spisuje się przy poprawianiu jakości przesyłanych danych, jednak też nie jest idealny. Przy średniej jakości kanału przepuszcza znikomą ilość błędnych ramek, co pozwala na dość wierne odtworzenie wtórnego obrazu.

Dla złej jakości kanału obraz dalej nie zostaje wyświetlony, jednak BER zostaje dzięki transmisji ARQ zmniejszony o ok. 4, 5 punktów procentowych, co jest dużo lepszym wynikiem niż przy kanale parity.

Warto zwrócić uwagę, że także przy tym testowaniu okazało się jak ważne jest dobranie mniejszej ramki, gdyż wtedy wykrycie błędu jest łatwiejsze, dzięki czemu poprawność kanału wzrasta.

Przykładowe rezultaty:

ARQ

Wybór Kanału

Modulo

Jakość kanału

Dobry

Ilość dopuszczalnych błędów

0

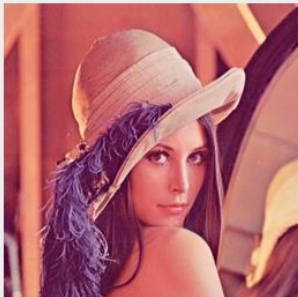
Ramka


8


Source image

Zdjęcie bez transmisji ARQ

Zdjęcie z transmisją ARQ







START

BEZ ARQ 0.005062317123793903%

ARQ 0.0030373902742763417%

Rys. 5 Dla kanału Modulo, podobnie jak dla Parity, możliwe jest wystąpienie i przepuszczenie przekłamanych informacji nawet przy najlepszej jakości kanału i małej ramce.

ARQ

Wybór Kanału

Modulo

Jakość kanału

Średni

Ilość dopuszczalnych błędów

5


Ramka


8


Source image

Zdjęcie bez transmisji ARQ

Zdjęcie z transmisją ARQ





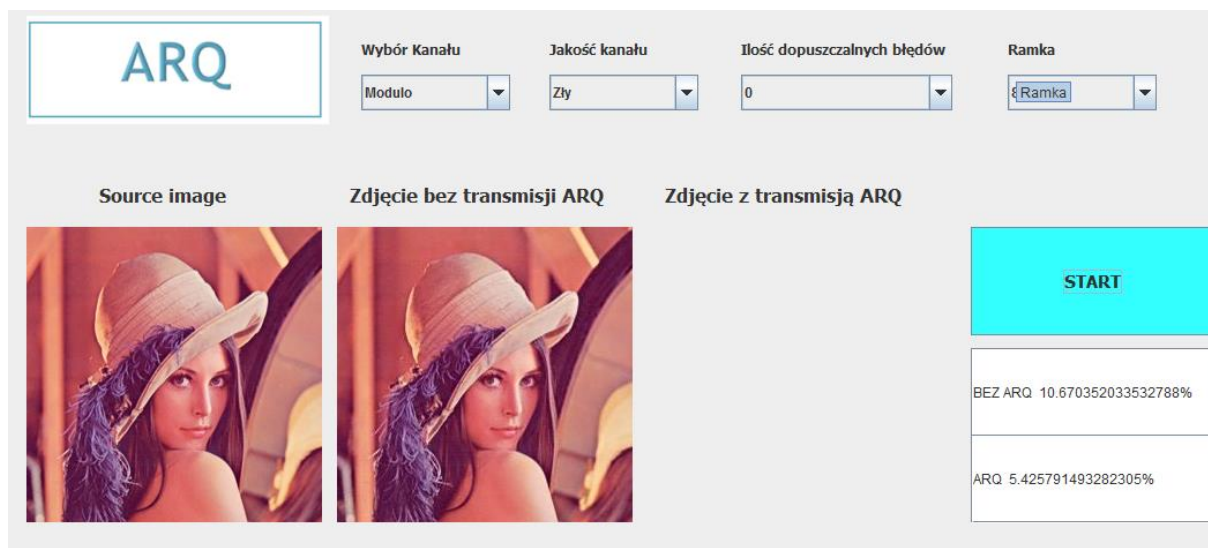


START

BEZ ARQ 0.06783504945883831%

ARQ 0.03442375644179854%

Rys. 6 Gdy przekłamate dane okażą się odpowiedzialne za kolor danego miejsca zdjęcia, wówczas nawet przy niskiej ilości dopuszczalnych błędów, może ono wyglądać zupełnie inaczej.

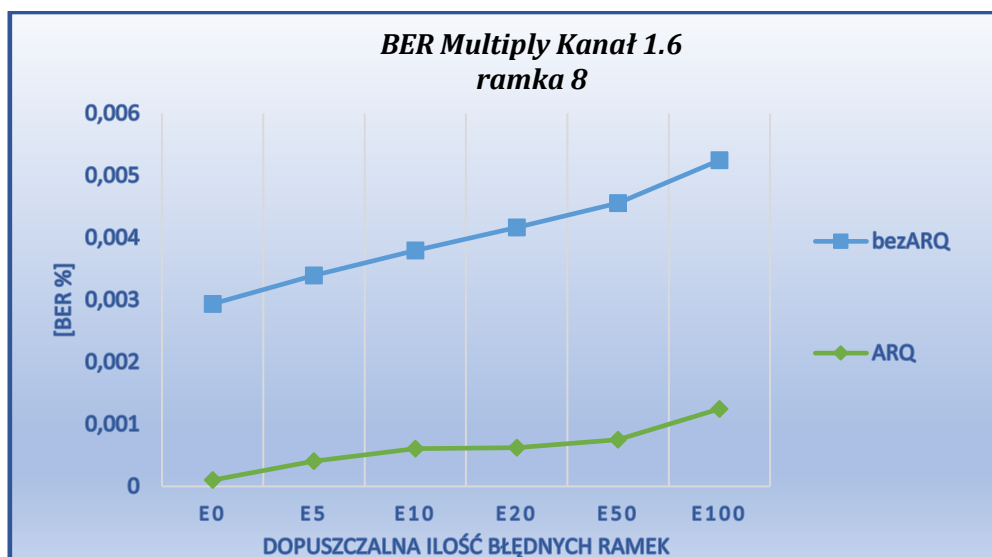


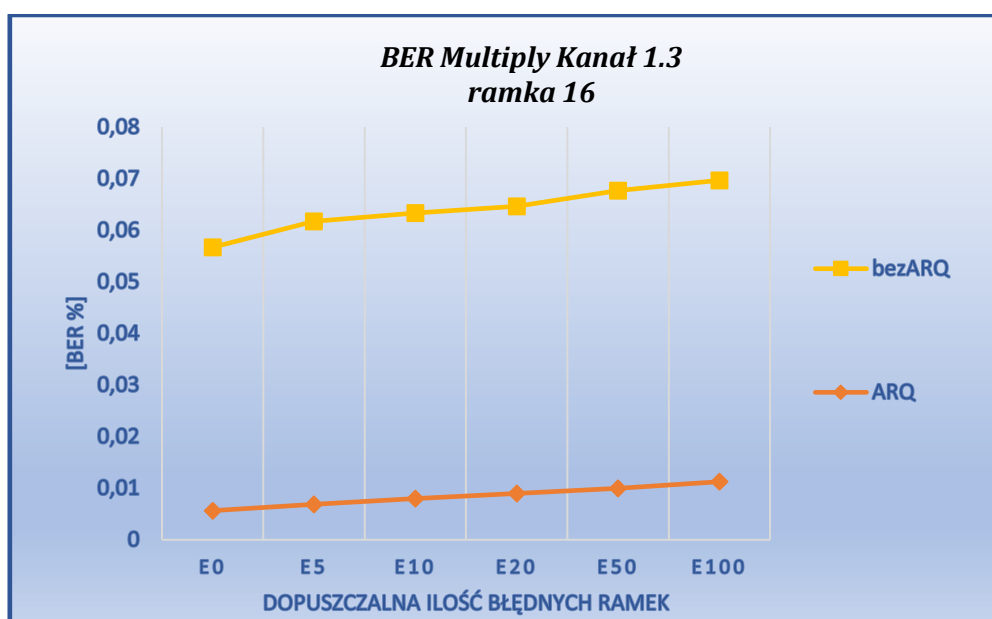
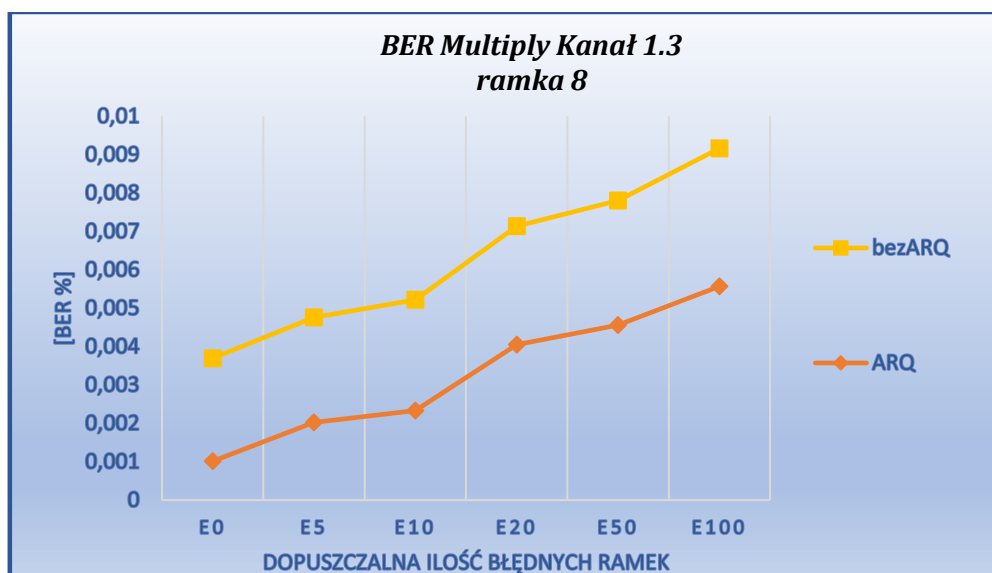
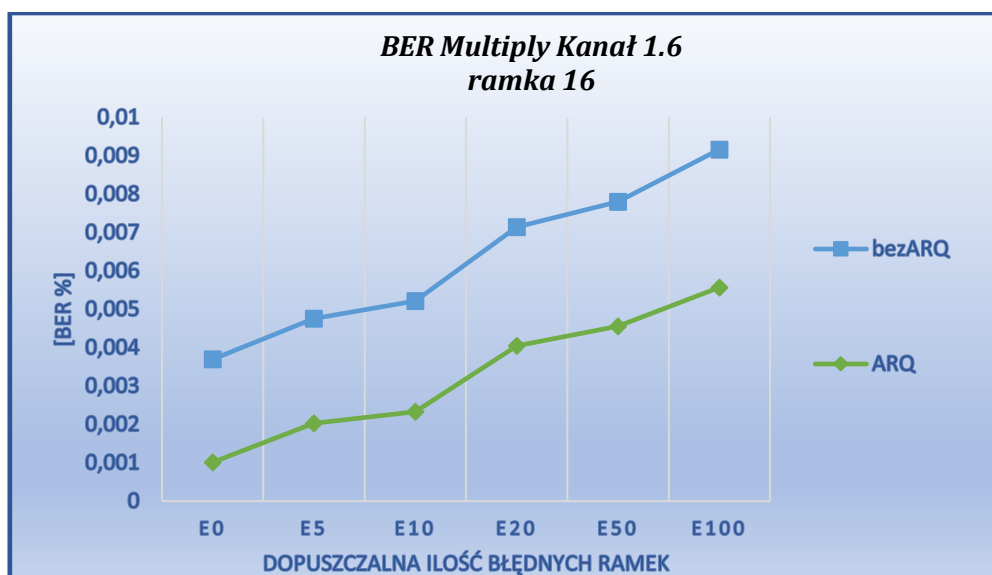
Rys. 7 Dla złego kanału, także dla metody modulo zdjęcia nie udało się odtworzyć

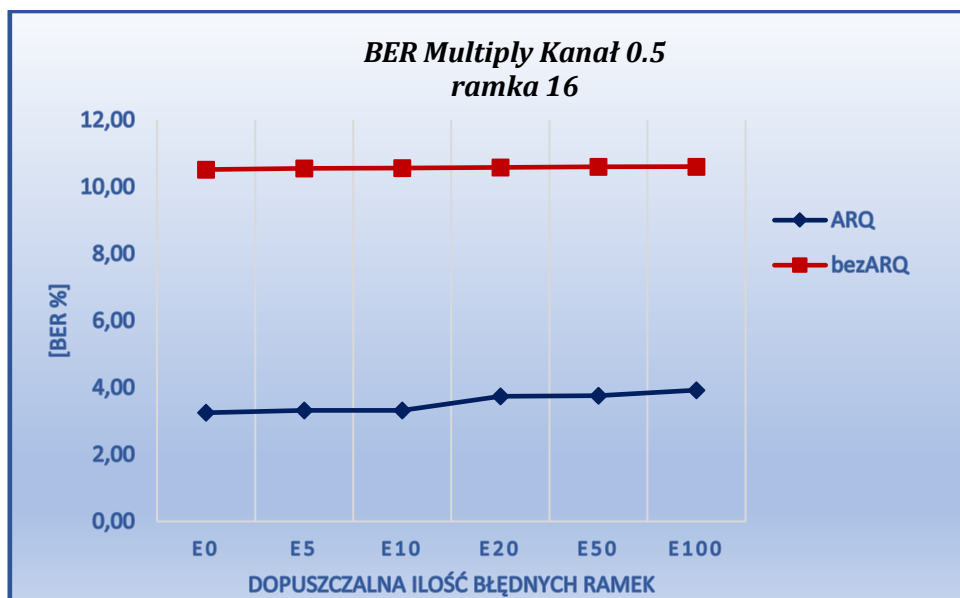
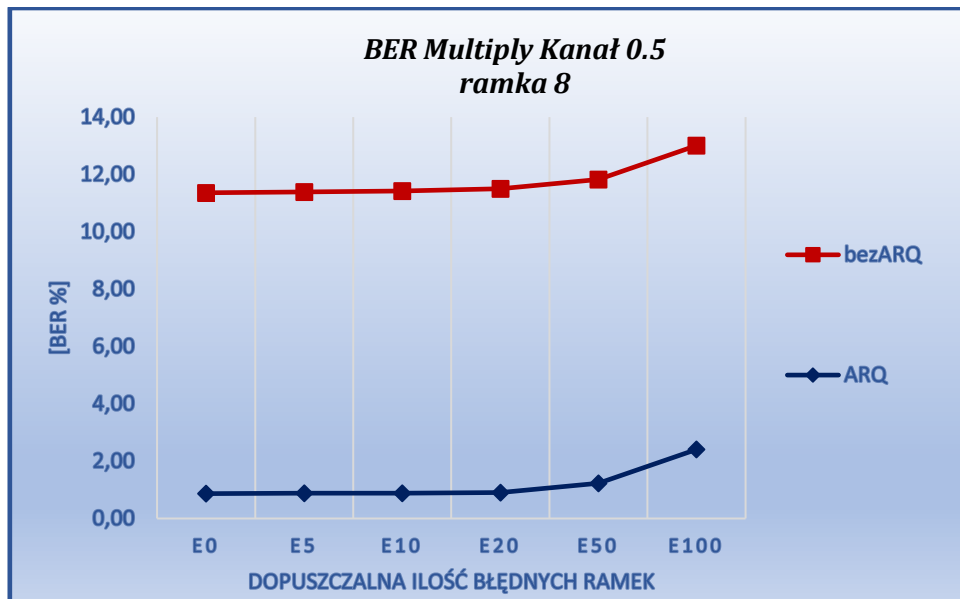
Kanał Multiply

Wybrane wyniki testowań:

Testowanie wielkości BER dla zwiększającej się ilość dopuszczalnych błędów







Wnioski:

Poniższe wyniki pokazują, że najlepiej działającym i efektywnym kanałem jest użyty przez nas kanał multiply. Pozwalał on na zmniejszenie ilości BER nawet przy bardzo słabym kanale do około 3%, co nie zostało osiągnięte przy użyciu wcześniejszych dwóch algorytmów. Rozmiar ramki także tutaj miał duży wpływ na jakość odbieranych danych.

Zauważono jednak, że przy słabej i średniej jakości kanałów pojawia się dużo błędów związanych z samym rozszerzeniem pliku (JPG), które mimo zminimalizowania ilości przekłamaných bitów nie pozwalały na odtworzenie obrazu.

Warto zwrócić uwagę na to, że przy kanale multiply wielkości BER dla dobrej jakości kanału są znacznie mniejsze w porównaniu do poprzednio testowanych algorytmów i oscylują w okolicach tysięcznych części procenta, co naprawdę jest niewielkim i satysfakcjonującym wynikiem. Mimo wszystko rozszerzenie JPEG jest dość trudnym typem plików do poprawnego przesłania, gdyż nawet jeden drobny błąd, ale w części przechowującej ważne dane o obrazie, może spowodować odebranie obrazu, który nie będzie zdalny do użytku, albo po prostu o niesatysfakcjonującej użytkownika jakości.

Przykładowe rezultaty:

ARQ

Wybór Kanału: Multiply | Jakość kanału: Dobry | Ilość dopuszczalnych błędów: 0 | Ramka: 16

Source image | Zdjęcie bez transmisji ARQ | Zdjęcie z transmisją ARQ

START

BEZ ARQ 0.006074780548552683%

ARQ 0.002024926849517561%

Rys. 8 Wielkość ramki bardzo wpływa na poprawność przesłanego zdjęcia, jednak dobry kanał oraz niska ilość dopuszczonych błędów pozwalają na prawie idealne poprawienie zdjęcia.

ARQ

Wybór Kanału: Multiply | Jakość kanału: Średni | Ilość dopuszczalnych błędów: 20 | Ramka: 8

Source image | Zdjęcie bez transmisji ARQ | Zdjęcie z transmisją ARQ

START

BEZ ARQ 0.05669795178649171%

ARQ 0.0%

Rys. 9 Kanał multiply działa o wiele lepiej dla średniego kanału i dopuszczalnej ilości błędów – widać wyraźnie, że potrafił zmniejszyć BER nawet do 0%.

ARQ

Wybór Kanału: Multiply | Jakość kanału: Zły | Ilość dopuszczalnych błędów: 5 | Ramka: 8

Source image | Zdjęcie bez transmisji ARQ | Zdjęcie z transmisją ARQ

START

BEZ ARQ 10.572143081331188%

ARQ 0.8484443499478581%

Rys. 10 Kanał multiply potrafi zminimalizować BER do wyniku, który pozwoliłby uzyskać odpowiednią jakość zdjęcia i je wyświetlić. Na powyższym zdjęciu nie udało się wyświetlić zdjęcia, ponieważ wystąpił błąd formatu JPG.

Przykładowe błędy napotkane w trakcie testowania programu:

- *Bogus marker length*
- *Bogus DQT index 4*
- *Bogus Huffman table definition*

Błąd bitowy w skanerze JPEG (kolor- błąd)

Jednym z najgorszych problemów z obrazami są pojedyncze lub występujące w dużych ilościach błędy bitów i bajtów w skanowaniu JPEG-a. Z powodu problemów z nośnikiem pamięci ewentualnie z powodu błędów transmisji niektóre fragmenty danych na obrazie są nieprawidłowe.

Takie przekłamania są szczególnie trudne dla odczytu, ponieważ format JPEG nie "wybacza" jakichkolwiek błędów (przynajmniej w podstawowej wersji JPEG, tak zwanej linii bazowej). Skanowanie JPEG ma następującą kompresję: Oznacza to, że region danych w pliku obrazu jest kompresowany za pomocą danych z poprzedniego obszaru danych w pliku. Jeśli pewna część danych w skanerze JPEG jest nieprawidłowa, dekodowanie spowoduje błędne dane dla tego samego regionu danych i dla wszystkich kolejnych regionów danych w pliku zdjęć. Ponieważ części obrazu są kodowane poziomo, pojawiają się tak obawy wyblakłe linie w złym kolorze. Obraz po prawej pokazuje typowy plik JPG z takim błędem.



Uszkodzony nagłówek JPG

Plik JPG można podzielić na dwie części.

Nagłówek JPEG (~ 0.05 procent)

JPEG-Scan (~ 99,95 procent)

- Jeśli nagłówek pliku obrazu jest nieprawidłowy, przywrócenie jest często niemożliwe.
- Nagłówek zdjęcia zawiera wszystkie dane niezbędne do dekodowania skanowania JPG.
- Nagłówek JPEG jest zbudowany w taki sposób, że nawet mały błąd sprawia, że cały nagłówek jest nieprawidłowy.

Wnioski:

Podsumowując wnioski zawarte przy testowaniu: do udanej transmisji najlepiej wykorzystać jak najlepszą jakość kanału przy małej ramce. Jeśli musimy skorzystać ze słabszej jakości kanału o większej ilości zakłóceń i błędów, warto użyć wydajniejszego algorytmu, który wychwyci więcej błędów – przy naszym testowaniu bezsprzecznie okazało się, że najlepszym algorytmem jest algorytm Multiply.

Testowanie plików JPG okazało się trudnym zadaniem, ze względu na specyficzną budowę tego pliku. Dane nagłówka zawierają bowiem najwięcej technicznych informacji o pliku, przez co ich utrata często powoduje niemożność ponownego odtworzenia pliku. Gdy nagłówek pliku JPG jest uszkodzony, nadal można używać nagłówka innego pliku zdjęć, zamiast oryginalnego nagłówka, pod warunkiem, że pliki JPG są technicznie podobne.