Transmisja w systemie Hybrid ARQ (HARQ)

1. Cel projektu

Celem projektu jest implementacja kanału komunikacyjnego opartego na modelach Binary Symmetric Channel (BSC) i Gilberta-Elliotta oraz systemu transmisji Hybrid Automatic Repeat Request (HARQ) z wykorzystaniem kodów detekcyjnych i korekcyjnych. Następnie przeprowadzenie symulacyjnej oceny skuteczności transmisji dla różnych parametrów kanałów i różnych sposobów transmisji.

1. Opis projektu

W ramach projektu planujemy stworzyć program symulujący działanie systemu transmisji HARQ, który będzie umożliwiał modyfikowanie różnych zmiennych w projekcie, takich jak:

* Parametry kanałów: planujemy zaimplementować modele BSC i Gilberta-Elliotta, których parametry będą mogły być modyfikowane przez użytkownika, takie jak prawdopodobieństwo błędu bitowego, prawdopodobieństwo przejścia kanału z dobrego na zły oraz zły na dobry.
* Kod detekcyjny: wykorzystamy kilka kodów detekcyjnych – bit parzystości, CRC8, CRC16, CRC32 które pozwolą na wykrycie błędów w przesyłanych danych.
* Kod korekcyjny: do korekcji błędów wykorzystamy kod RS, który pozwoli na korekcję błędów w przesyłanych danych.

Przetestujemy Typ II Hybrid ARQ:

* W Typie II Hybrid ARQ pierwsza wiadomość jest kodowana tylko kodem detekcyjnym. Jeśli pierwsza transmisja zostanie odebrana bez błędów, jest to koniec. Natomiast jeśli wykryto błędy dosyłana jest część korekcyjna.

1. Planowane efekty projektu

Poza implementacją systemu transmisji HARQ, planujemy przeprowadzić symulacyjne badania, które pozwolą nam ocenić skuteczność transmisji dla różnych parametrów kanałów i różnych sposobów transmisji. W przypadku testów, planujemy wykorzystać modele kanału z różnymi parametrami, takimi jak:

* Prawdopodobieństwo błędu bitowego
* Prawdopodobieństwo przejścia kanału ze stanu dobrego na zły oraz zły na dobry
* Długość przesyłanej wiadomości
* Liczba potrzebnych retransmisji

Wyniki badań pozwolą nam ocenić skuteczność systemu transmisji HARQ dla różnych przypadków, co pozwoli nam określić optymalne parametry systemu transmisji dla danego kanału.

1. Źródła literaturowe

Do realizacji projektu planujemy skorzystać z różnych źródeł, w tym z artykułów naukowych, materiałów internetowych oraz filmów na platformach takich jak YouTube. W artykułach naukowych będziemy szukać informacji dotyczących teorii kodowania, metod detekcji i korekcji błędów, a także sposobów projektowania systemów komunikacyjnych.

Do tej pory, aby zdobyć informację korzystaliśmy z następujących źródeł:

<https://devopedia.org/hybrid-arq>

<https://en.wikipedia.org/wiki/Hybrid_automatic_repeat_request>

<https://www.youtube.com/watch?v=MN8KIdGYnic&ab_channel=FOREACH>

<https://www.youtube.com/watch?v=1pQJkt7-R4Q&ab_channel=vcubingx>

<https://www.youtube.com/watch?v=fBRMaEAFLE0&t=584s&ab_channel=Computerphile>

<https://www.youtube.com/watch?v=aSqGKNRJRDg&ab_channel=KhanAcademyPoPolsku>

1. Implementacja

Zdecydowaliśmy się na zaimplementowanie naszego systemu transmisji w języku programowania Java. Projekt jest aktualnie dostępny na repozytorium github i na bieżąco odświeżany: <https://github.com/radziu2402/HybridARQ-NIDUC2-projekt>

Program pobiera zdjęcie w formacie BMP (wybór tego formatu pozwala na mniejsze problemy przy ponownym odtwarzaniu obrazka z przesłanych bitów), a następnie dzieli je na paczki i po kolei przesyła, jeśli wiadomość zakodowana detekcyjnie przejdzie bezbłędnie przez kanał to przechodzimy do kolejnej paczki, ale jeśli wykryty zostanie błąd to dosyłana jest część korekcyjna i która próbuje naprawić błędy, a jeśli jej się nie uda to następuje retransmisja, aż do skutku.

1. Zakres testów

Mamy do dyspozycji wiele parametrów, którymi możemy manipulować, testy są przeprowadzane z podziałem na:

1. Rodzaj wybranego kanału BSC i Gilberta-Elliotta
2. Prawdopodobieństwo błędu w tym kanale
3. Rodzaj wybranego kodu korekcyjnego (niektóre przepuszczają więcej błędów co powoduje zmienione piksele w końcowym obrazie)
4. Długość jednej paczki bitów (musimy się jednak ograniczyć do wielokrotności 1 bajta, czyli 8,16,24 bity i tak dalej z uwagi na implementację kodu RS)
5. Długość części korekcyjnej kodu RS, tutaj również są to wielokrotności 1 bajta

**Zaczęliśmy od przeprowadzania testów dla kanału BSC:**

Aby obrazek w ogóle został wygenerowany prawdopodobieństwo błędu musi być dość niskie dlatego nasze badania przeprowadzane są dla kilku wartości:

* 0,01%
* 0,05%
* 0,1%

**Parametry:**

Długość pakietu 1 bajt, długość części korekcyjnej 3 bajty

**Kod detekcyjny – bit parzystości**

**Prawdopodobieństwo błędu 0,01%:**

Liczba jeden oznacza ze pakiet przeszedł bezbłędnie lub błędy nie zostały po prostu wykryte. Natomiast liczba 2 oznacza ze wystarczyło dosłać część korekcyjna. Następne liczby to już kolejne retransmisje tego pakietu.

Liczba pojedynczych bitów przepuszczonych mimo błędu: 4107



Rysunek 1 Obrazek wyjściowy

Jak widać bit parzystości bardzo słabo zabezpiecza naszą transmisję mimo, że szanse na przekłamanie bitu wynosiły tylko 0,01%.

**Prawdopodobieństwo błędu 0,05%:**

W tym przypadku obrazu nie udało się nawet utworzyć, ponieważ błędy pojawiły się w bajtach odpowiadających kodowaniu rozszerzenia pliku.

Liczba pojedynczych bitów przepuszczonych mimo błędu: 20439

**Prawdopodobieństwo błędu 0,1%:**

Analogicznie obrazu nie udało się nawet utworzyć z powodu zbyt wielu błędów.

Liczba pojedynczych bitów przepuszczonych mimo błędu: 40963

**Kod detekcyjny – CRC8**

**Prawdopodobieństwo błędu 0,01%:**

Liczba pojedynczych bitów przepuszczonych mimo błędu: 4

**Prawdopodobieństwo błędu 0,01%:**

Liczba pojedynczych bitów przepuszczonych mimo błędu: 294

**Prawdopodobieństwo błędu 0,1%:**

Liczba pojedynczych bitów przepuszczonych mimo błędu: 1257

**Kod detekcyjny – CRC16**

**Prawdopodobieństwo błędu 0,01%:**

Liczba pojedynczych bitów przepuszczonych mimo błędu: 0

**Prawdopodobieństwo błędu 0,05%:**

Liczba pojedynczych bitów przepuszczonych mimo błędu: 9



Rysunek Obrazek wyjściowy

**Prawdopodobieństwo błędu 0,1%:**

Liczba pojedynczych bitów przepuszczonych mimo błędu: 87

Jak widać kod CRC16 już bardzo skutecznie daje sobie rade z wykrywaniem błędów,

**Kod detekcyjny – CRC32**

**Prawdopodobieństwo błędu 0,01%:**

Liczba pojedynczych bitów przepuszczonych mimo błędu: 0

**Prawdopodobieństwo błędu 0,05%:**

Liczba pojedynczych bitów przepuszczonych mimo błędu: 7

**Prawdopodobieństwo błędu 0,1%:**

Z badań wprost wynika, że im dłuższy kod detekcyjny przyjmiemy to nasze dane będą lepiej zabezpieczone mimo, a im większe mamy prawdopodobieństwo błędu w kanale tym proporcjonalnie więcej retransmisji potrzebujemy. Jak do tej pory jednak żaden z kodów, nawet najdłuższy CRC32, nie zdołał wychwycić każdego błędu przy prawdopodobieństwach 0,5% i 0,1%.

Tabela Liczba przepuszczonych błędów

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bit parzystości | CRC8 | CRC16 | CRC32 |
| 0,01% | 4107 | 4 | 0 | 0 |
| 0,05% | 20439 | 294 | 9 | 7 |
| 0,10% | 40963 | 1257 | 87 | 80 |

Zależność ta jest odpowiada zależności liniowej.

Następnie po konsultacji z prowadzącym dokonaliśmy wydłużenia naszych pakietów i znowu dokonaliśmy pomiarów.

**Parametry:**

Długość pakietu 16 bajtów, długość części korekcyjnej 6 bajty

**Kod detekcyjny – bit parzystości**

**Prawdopodobieństwo błędu 0,01%:**

Liczba pakietów przepuszczonych mimo błędu: 1640

**Prawdopodobieństwo błędu 0,03%**

Liczba pakietów przepuszczonych mimo błędu: 2973

**Prawdopodobieństwo błędu 0,05%:**

Liczba pakietów przepuszczonych mimo błędu: 3120

W tym momencie możemy już zauważyć, że tak dobrane parametry czyli: 16 bajtowa wiadomość i tylko 6 bajtowa część korekcyjna nie radzi sobie z naprawą błędów przy prawdopodobieństwie 0,05% i na pewno jest to nieefektywna transmisja.

**Kod detekcyjny – CRC8**

**Prawdopodobieństwo błędu 0,01%:**

Liczba pakietów przepuszczonych mimo błędu: 140

**Prawdopodobieństwo błędu 0,03%:**

Liczba pakietów przepuszczonych mimo błędu: 353

**Prawdopodobieństwo błędu 0,05%:**

Liczba pakietów przepuszczonych mimo błędu: 643

**Kod detekcyjny – CRC16**

**Prawdopodobieństwo błędu 0,01%:**

Liczba pakietów przepuszczonych mimo błędu: 0

**Prawdopodobieństwo błędu 0,03%:**

Liczba pakietów przepuszczonych mimo błędu: 2

**Prawdopodobieństwo błędu 0,05%:** Liczba pakietów przepuszczonych mimo błędu: 32

**Kod detekcyjny – CRC32**

**Prawdopodobieństwo błędu 0,01%:** Liczba pakietów przepuszczonych mimo błędu: 0

**Prawdopodobieństwo błędu 0,03%:** Liczba pakietów przepuszczonych mimo błędu: 1

**Prawdopodobieństwo błędu 0,05%:** Liczba pakietów przepuszczonych mimo błędu: 23

Tabela 2 Liczba przepuszczonych pakietów z błędami

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bit parzystości | CRC8 | CRC16 | CRC32 |
| 0,01% | 4107 | 140 | 0 | 0 |
| 0,03% | 20439 | 353 | 2 | 1 |
| 0,05% | 40963 | 643 | 15 | 17 |