Niezawodność i diagnostyka układów cyfrowych 2
System ARQ (Automatic Repeat Request)

Chrzanowski Damian 247935
Fedorczak Paweł 243751

Grupa: Środa, 7^{30}-9^{00} TN
Semestr letni 2019/2020

Spis treści

1 Część teoretyczna **3**
1.1 Temat projektu . . . 3
1.2 Użyte algorytmy . . . 3
1.2.1 Bit parzystości . . . 3
1.2.2 Wzdłużna kontrola nadmiarowa . . . 3
1.2.3 Algorytm Verhoeffa . . . 3

2 Część praktyczna **4**
2.1 Symulator . . . 4
2.1.1 Opcje symulatora . . . 4

3 Wyniki symulacji **5**
3.1 Stopa błędów . . . 5
3.1.1 Kanał BSC (pojedyncze błędy) . . . 5
3.1.2 Kanał BURST (grupowe błędy) . . . 5
3.2 Efektywność . . . 5
3.2.1 Kanał BSC (pojedyncze błędy) . . . 5
3.2.2 Kanał BURST (grupowe błędy) . . . 5
3.3 Nadmiarowość . . . 5
3.4 Analiza wyników . . . 5

4 Podsumowanie **6**

5 Bibliografia **6**

1 Część teoretyczna

1.1 Temat projektu

Nasza grupa projektowa wybrała temat sytemu ARQ. System ARQ jest systemem kontroli błędów w telekomunikacji. System działa na zasadzie używania potwierdzeń przez odbiornik, który wysyła potwierdzenie odebrania pakietu danych w stanie dobrym (przez co następuje wysłanie kolejnej paczki danych) albo złym (gdzie następuje ponowne wysłanie paczki danych). To czy pakiet zostanie odebrany pozytywnie zależy od wyniku testu sumy kontrolnej. Dodatkowo system może też patrzeć na czas wysłania pakietu danych, jeżeli pakiet nie zostanie wysłany w całości w danym przedziale czasowym odbiornik wysyła wiadomość zwrotną, która informuje o ponownym nadaniu pakietu.

System ARQ używa różnych protokołów i algorytmów sum kontrolnych w zależności od rodzaju i wielkości wysyłanych danych. Dwa podstawowe protokoły systemu ARQ to stop-and-wait oraz go-back-n.

- Stop-and-wait - odbiornik na bieżąco odbiera pakiety, a następnie oblicza sumę kontrolną po czym wysyła wiadomość zwrotną do nadajnika, który na podstawie odpowiedzi wysyła kolejną paczkę danych albo nadaje ponownie tę samą.
- Go-Back-N - odbiornik odbiera pakiety i przetrzymuje je w buforze, którego rozmiar określa się rozmiarem okna. Znaczącą zmianą w stosunku do poprzedniego protokołu jest to, że wysyłanie danych nie jest zatrzymywanie (o ile bufor nie zostanie zapełniony) i nadawca nie potrzebuje się wstrzymywać w wysyłaniu danych przez czekanie na odpowiedź. Natomiast minusem takiego protokołu jest potrzeba dodatkowej pamięci na przetrzymywanie danych.

1.2 Użyte algorytmy

1.2.1 Bit parzystości

Bit parzystości jest jednym z najprostszych algorytmów wykrywania przekłamania bitów w pakiecie danych. Sam algorytm polega na dopisaniu na końcu pakietu 0 w przypadku kiedy pakiet danych posiada w ciągu bitów parzystą liczbę 1. A 1 dopisujemy kiedy 1 w ciągu bitów jest nieparzysta liczba. Ze względu na prymitywność bit parzystości jest słabym algorytmem sumy kontrolnej. Przy szansie na przekłamanie bitów w słabych kanałach, gdzie dochodzi do zmiany kilku bitów w pakiecie może zdarzyć się sytuacja na przepuszczenie złego pakietu.

1.2.2 Wzdłużna kontrola nadmiarowa

Wzdłużna kontrola nadmiarowa jest rozszerzeniem powyższego algorytmu bitu parzystości. W LRC pakiet jest dzielony na n części. Z każdej takiej części jest brany po kolei każdy bit, a za pomocą utworzonego ciągu wyliczany jest bit parzystości. Po wyliczeniu każdego bitu parzystości powstaje kod lrc.

Na przykład możemy wziąć ciąg bitów 11100111 11011101 00111001 10101001. Taki ciąg dzielimy na cztery części. Teraz patrząc na pierwszy bit z każdej części wyliczamy bit parzystości (1101 daje bit równy 1), resztę bitów obliczamy tak samo. Kiedy wyliczymy wszystkie w ten sposób bity dostaniemy kod lrc.

	1	1	1	0	0	1	1	1
	1	1	0	1	1	1	0	1
	0	0	1	1	1	0	0	1
	1	0	1	0	1	0	0	1
lrc =	**1**	0	1	0	1	0	1	0

1.2.3 Algorytm Verhoeffa

Algorytm Verhoeffa, który służy do obliczania sumy kontrolnej pozwala na wykrycie

- wszystkich błędów na jednym znaku
- wszystkich błędów zamiany kolejności dwóch znaków
- większość podwójnych błędów
- większość podwójnych błędów oddzielonych znakiem
- większość zmian kolejności trzech znaków

w przypadku systemu ARQ interesuje nas wyłącznie wykrywanie tego pierwszego błędu. Sam algorytm polega na wykorzystaniu trzech tablic:

- tabelda **d** - przechowuje wartości mnożenia w grupie dihedrealnej D5

- tablica **p** zawiera permutację dla każdej cyfry, bazujące na jej pozycji w sprawdzanym identyfikatorze
- tablica
- inv przedstawia element odwrotny mnożenia w grupie D5 przedstawionego w pierwszej tabeli, czyli taki że: $d(i, inv(i)) = 0$

Suma kontrolna wyliczania jest w następujący sposób:

- tworzymy powyższe tablice pomocnicze
- inicjalizujemy zmienną cyfry kontrolnej $n = 0$
- inicjalizujemy zmienną pomocniczą $c = 0$
- dla każdego indeksu i z tablicy bitów wyliczamy c używając tablic $c = d(c,\ p(i \bmod 8,\ n_i))$
- po zakończeniu iteracji zamieniamy cyfrę kontrolną ze zmienną pomocniczą c $n_0 = inv(c)$

2 Część praktyczna

2.1 Symulator

Symulator został napisany w języku Python 3.7 bez wykorzystywania gotowych rozwiązań. Symulator składa się z modułów, a każdy moduł jest odpowiedzialny za inne działanie.

- Test.py - opcje symulatora, tworzenie zmiennych innych modułów
- Coder.py - wyliczanie i dodawanie sumy kontrolnej do pakietu danych, także sprawdzanie jej poprawności
- NoiseGenerator.py - odpowiada za wprowadzanie przekłamań w pakiecie danych podczas transmisji zgodnie z opcjami ustawionymi w Test.py
- BitPacket.py - moduł opakowujący pakiet danych, rozdziela część danych od sumy kontrolnej dla łatwiejszego wykonywania obliczeń
- ARQSystem.py - główny moduł odpowiedzialny za symulację. Dzieli się na dwie części, pierwsza jako cały system, a druga jako symulator kanału i protokołu.

2.1.1 Opcje symulatora

Symulator w pliku Test.py posiada na początku zmienne, które służą jako opcje symulatora. Nie wszystkie opcje na raz są wykorzystywane, zależy to od wybranego kanału - BSC albo BURST.

- number_of_packets - liczba pakietów
- packet_size - rozmiar pakietu
- chance_for_packet - szansa na to, że będzie istnieć szansa na przekłamanie pakietu
- chance_for_bit - szansa na przekłamanie bitu
- chance_to_series_of_errors - szansa na to, że będzie istnieć szansa na rozpoczęcie błędów grupowych
- chance_to_start_series_of_errors - szansa na rozpoczęcie błędów grupowych
- chance_to_end_series_of_errors - szansa na zakończenie błędów grupowych
- protocol_name - nazwa protokołu
- checksum_mode - alogrytm sumy kontrolnej
- channel - kanał, którym przesyłane są dane
- file_name - nazwa pliku, w którym zapisane są wyniki

3 Wyniki symulacji

3.1 Stopa błędów

3.1.1 Kanał BSC (pojedyncze błędy)

	Szansa na przekłamanie			
Algorytm	0.001%	0.0001%	0.00001%	0.000001%
PARITY BIT	1.5750287051589177e-05	1.0952186360376582e-06	1.0058130330958085e-07	1.1175700367731205e-08
LRC	2.8951189159292035e-05	1.4006236649374428e-06	1.639027693011901e-07	1.4900251754653646e-08
VERHOEFF CODE	1.8126156259536163e-05	1.1026186298443698e-06	8.567644758925847e-08	1.4900251754653646e-08

3.1.2 Kanał BURST (grupowe błędy)

	Szansa na przekłamanie			
Algorytm	0.001%	0.0001%	0.00001%	0.000001%
PARITY BIT	0.0013330524407968018	9.682999321947908e-05	1.195427416001648e-05	6.630915551520515e-07
LRC	0.002954302715898688	0.00011708617828806836	1.1294390830027464e-05	2.987500476808056e-06
VERHOEFF CODE	0.0018368285350549283	0.0001214184264857339	1.0575453682865426e-05	6.742363918980775e-07

3.2 Efektywność

3.2.1 Kanał BSC (pojedyncze błędy)

	Szansa na przekłamanie			
Algorytm	0.001%	0.0001%	0.00001%	0.000001%
PARITY BIT	0.6322939178758876	0.9377289377289377	0.9934513703613873	0.9992681141741888
LRC	0.4351891202719932	0.9183856502242153	0.9893719806763285	0.9990243902439024
VERHOEFF CODE	0.5385222193005522	0.9360146252285192	0.9944161204175771	0.9990243902439024

3.2.2 Kanał BURST (grupowe błędy)

	Szansa na przekłamanie			
Algorytm	0.001%	0.0001%	0.00001%	0.000001%
PARITY BIT	0.7562776957163959	0.9685504847481674	0.9968362131905573	0.9995119570522206
LRC	0.4327979712595097	0.927536231884058	0.9893719806763285	0.9980506822612085
VERHOEFF CODE	0.5571273122959739	0.9403122130394858	0.9941747572815534	0.9995119570522206

3.3 Nadmiarowość

Nadmiarowość jest dość łatwa do wyliczenia i zależy wyłączenie od użytego algorytmu wykrywania przekłamania pakietu.

- Bit parzystości - 1 dodatkowy bit na pakiet
- LRC - 16 384 dodatkowych bitów ze względu na podział pakietu na 4 części
- Algorytm Verhoeffa - 4 dodatkowe bity na pakiet

3.4 Analiza wyników

- Pomimo lepszej efektywności w kanale błędów grupowych samo BER jest większe
- Algorytm LRC w porównaniu do pozostałych algorytmów wymaga strasznie dużo liczenia, dlatego lepszym protokołem jest dla niego go-back-n, z drugiej strony wymaga także dodatkowej pamięci dla bufora w odbiorniku
- Bardzo dobre jakości kanałów tracą na znaczeniu w pewnym momencie (można tutaj spojrzeć na różnicę pomiędzy kanałem z szansą na przekłamanie na poziomie 0.001% i 0.0001% a 0.00001% i 0.000001%) W takich przypadkach lepiej wybrać algorytmy niewymagające dużej liczby obliczeń lub/i dodatkowej ilości danych

4 Podsumowanie

W ramach projektu zapoznaliśmy się z działaniem systemu ARQ, podstawowymi protokołami i algorytmami stosowanymi w telekomunikacji w systemie automatycznego żądania powtórzenia. Przez napisanie symulatora od podstaw również z działaniem symulacji „od środka",choć z drugiej strony przez brak optymalizacji dla symulacji takiego systemu czas wykonywania obliczeń sięgał nawet kilkudziesięciu minut. Zrozumieliśmy jak jakoś kanału wpływa na BER oraz efektywność, zwłaszcza jak dużo błędów istnieje w słabej jakości kanałów (szansa na przekłamanie 0.0001% i więcej).

5 Bibliografia

- Materiały na stronie Zakładów Systemów Komputerowych i Dyskretnych PWr dotyczące wykładu
- Materiały na stronie Zakładów Systemów Komputerowych i Dyskretnych PWr dotyczące projektu
- Error Detection and Correction
- Podstawowe pojęcia. Techniki wykrywania błędów
- Podstawowe modele kanałów telekomunikacyjnych
- Algorytm Verhoeffa