Michał Budnik – Sprawozdanie 3

# 1. Zadanie 1

## 1.1 Opis zagadnienia

Zadanie 1 polega na zaimplementowaniu ramkowania zgodnie z zasadą “rozpychania bitów” oraz weryfikacji poprawności przesłanych danych metodą CRC. Program ma odczytywać plik ‘Z.txt.’ zawierający dowolny ciąg składający się ze znaków ‘0’ i ‘1’, następnie zapisywać ramkami odpowiednio sformatowany ciąg do pliku ‘W.txt’. Program ma również obliczać i wstawiać do ramki pola kontrolne CRC. Dodatkowo należy napisać program wykonujący procedurę odwrotną – mając plik zawierający ramkowany ciąg bitów należy sprawdzić poprawność przesłanych danych oraz deramkuje dane.

## 1.2 Założenia

W celu napisania tych programów zostały użyte następujące założenia:

- wykorzystywany zostaje format CRC16,  
- kod CRC jest sprawdzany i dodawany do kodu przed ramkowaniem – w przypadku CRC16, który składa się z 16 bitów, jest to niezbędne w celu ominięcia przedwczesnego końca transmisji. Wiąże się to jednak z kosztem – wielkość pakietu nie jest stała, zależy ona od ilości rozepchanych bitów,  
- flaga ramkowania została ustalona jako ‘01111110’, przy czym SOH to ‘01111110’ a EOT to ‘01111111’,  
- programy są napisane w języku Python w celu prostego przedstawienia rozwiązania problemu.

## 1.3 Realizacja

Wczytywany zostaje symulowany ciąg bitów z pliku ‘Z.txt’. Ciąg bitów zostaje następnie dzielony na 32-bitowe części, dzięki temu jeden pakiet osiąga średnią wielkość 70-bitów. Używając biblioteki crcmod, dla każdej części zostaje wyliczana jej wartość CRC16, po czym jest ona konwertowana z systemu hexadecymalnego do binarnego, uzupełniana o odpowiednią ilość poprzedzających zer, aby długość kodu CRC zawsze wynosiła 16 bitów, oraz dopisywana na koniec danej części transmisji.   
Następnie każdy 48-bitowy ciąg zostaje ramkowany poprzez rozepchnięcie bitowe. Zaimplementowany algorytm iteruje po bitach danego ciągu. Gdy napotka bit ‘0’ ustawia flagę rozpoczęcia ciągu na wartość True, oraz ustawia wartość licznika na 1. Kolejno iterując sprawdza, czy następne bity w ciągu odpowiadają bitom flagi ramkowania. Gdy sześć kolejnych bitów sobie odpowiada, ciąg zostaje rozepchnięty o bit ‘0’, przykład:

Ciąg: 001111101  
Flaga ramkowania: 01111110  
Rozepchnięty ciąg: 0011111001

Gwarantuje to, że ciąg składający się z zera i sześciu następujących po sobie jedynek nigdy się nie powtórzy w bitach przeznaczonych do transmisji. Dzięki temu możemy wstawić dwa specjalne wyrazy aby jednoznacznie wskazać początek i koniec transmisji. Końcowa transmisja:

01111110 0011111001 01111111  
SOH Ramkowany\_ciąg EOT

Tak sformatowane ciągi zostają znów połączone w jeden ciąg oraz zapisane do pliku ‘W.txt’.

Program realizujący procedurę odwrotną opiera się na tych samych algorytmach. Wczytuje on plik ‘W.txt’, rozdziela na odpowiednie transmisje przeszukując występowanie znaku końca transmisji, usuwa SOH oraz EOT, oraz taką samą metodą jak pierwszy program usuwa nadmiarowe zera. Następnie odcina ostatnie 16 bitów nieramkowanej już transmisji, sprawdza CRC16 pozostałego ciągu, porównuje nowo powstały CRC z odciętymi 16 bitami. Jeżeli są one równe to sygnał dotarł bezproblemowo, jeżeli natomiast się różnią to sygnał został w którymś miejscu zmieniony. Następnie konkatenuje ze sobą wszystkie ciągi przez co, przy założeniu pełnego otrzymania poprawnej transmisji, uzyskiwany jest dokładnie taki sam ciąg jak w pliku ‘Z.txt’.

def destuffing(data, flag):

temp = data.replace(flag, '')

newdata = temp.split(flag[:-1] + '1')[:-1]

for j in range(0, len(newdata)):

iterFlag = False

count = 0

i = 0

while i < len(newdata[j]):

if (not iterFlag) and newdata[j][i] == flag[count]:

iterFlag = True

count = 1

elif iterFlag and count == 6 and newdata[j][i] == '0':

count = 1

newdata[j] = newdata[j][:i] + newdata[j][i+1:]

i-=1

elif iterFlag and count < 6 and newdata[j][i] == flag[count]:

count+=1

else:

count = 0

iterFlag = False

if newdata[j][i] == '0':

i-=1

i+=1

return newdata

Kod 1: Funkcja zdejmująca ramkę i usuwająca nadmiarowe '0' z ciągu.

## 1.4 Wnioski

Ramkowanie poprzez rozpychanie bitów wydaje się być bardzo dobrą metodą na przekazywanie danych w pakietach – w prosty, zrozumiały, oraz jednoznaczny sposób przetwarza ciąg bitów w taki sposób, aby odbiorca mógł łatwo określić gdzie jest początek a gdzie koniec transmisji. Metoda sprawdzania poprawności kodu CRC również wydaje się logiczną i jednoznaczną metodą na sprawdzenie poprawności przesyłanej transmisji. Zaimplementowanie obu tych rozwiązań w sposób opisany w tym sprawozdaniu wiąże się jednak z kilkoma usprawnieniami:

- CRC jest automatycznie przesyłane w tym samym pakiecie w celu weryfikacji poprawności danych,  
- pakiety są zawsze długości większej niż 63 bity i mniejszej niż 76 bitów,  
- metody użyte do ramkowania, rozepchnięcia bitów i wyliczenia CRC są praktycznie takie same jak metody do zdjęcia ramki, usunięcia nadmiarowych ‘0’ i sprawdzenia CRC,

- istnieje bardzo mała, lecz niezerowa, szansa na zniekształcenie transmisji w taki sposób, że SOH i EOT pozostają poprawne, oraz CRC wskazuje, że transmisja została przesłana poprawnie,  
- podatność na ataki typu man-in-the-middle – łatwość wychwycenia sygnału i przesłania innego spełniającego warunki poprawności dla odbiorcy

# 2. Zadanie 2

## 2.1 Opis zagadnienia

Zadanie 2 polega na napisaniu programu symulującego Ethernetową metodę dostępu do medium transmisyjnego CSMA/CD. Wspólne łącze realizowane jest za pomocą tablicy propagacja sygnału symulowana jest za pomocą propagacji wartości do sąsiednich komórek.

## 2.2 Założenia

W celu napisania tego programu zostały użyte następujące założenia:

- długość medium może być zmieniana,  
- sprawdzane jest przesyłanie danych między dwoma klientami – jednym znajdującym się na początku medium, drugim znajdującym się na końcu medium,  
- pakiety wysyłane przez klientów mogą być długości od 2 do 16 bitów,  
- w przypadku gdy klient widzi, że na jego końcu nie ma sygnału zaczyna on transmitować pakiet,  
- w przeciwnym przypadku czeka on na puste łącze,  
- w przypadku kolizji transmisja nie zostaje powtórzona, a odpowiednia ilość zniekształconych danych dalej ma szansę dotarcia do celu.

## 2.3 Realizacja

Utworzona została klasa Ethernet która symuluje wszystkie wymagania zadania. Znajdują się w niej informacje o długości medium, statystyki reprezentujące ilość kolizji pakietów oraz ilość poprawnie przesłanych pakietów, oraz metody potrzebne do symulacji połączenia Ethernetowego oraz wysyłania pakietów.

Dla obydwu klientów tworzone są losowo tablice zawierające sygnał ‘1’ lub ‘0’. Zarówno w tablicach jak i w symulacji kabla ‘1’ oznacza pakiet a ‘0’ oznacza jego brak. W każdym z symulowanych cykli istnieje prawdopodobieństwo p, że klient będzie chciał wysłać pakiet. Każdy pakiet ma losowo przypisywaną długość od 2 do 16 bitów. W ten sposób powstają dwie tablice – ‘rInject’ oraz ‘lInject’, które pokazują czy w danym cyklu klient chce wprowadzić sygnał do medium komunikacyjnego czy nie.

Następnie odbywa się symulacja n cyklów. Każdy cykl dzieli się na dwa zadania:

1. Przesunięcie pakietów i sprawdzenie kolizji,
2. Próba wprowadzenia pakietów do medium.

Realizacja pierwszego zadania jest trywialna, gdyż przy wprowadzaniu pakietów podawana jest dodatkowa flaga mówiąca w którym kierunku pakiet ma się poruszać. Jeżeli pakiet chce wejść na pole, które jest zajęte przez pakiet nadciągający z przeciwnej strony, następuje kolizja, inkrementowany jest licznik kolizji, a pola na których znajdują się te kolidujące pakiety są ustawiane na ‘0’. Jeżeli pakietowi natomiast udało się dotrzeć do końca kabla inkrementowany jest licznik poprawnie przesłanych pakietów.

Próba wprowadzenia pakietów do medium odbywa się w pierwszej kolejności poprzez sprawdzenie czy klient chce cokolwiek przesłać. W tym celu sprawdzany jest pierwszy element tablicy ‘lInject’ (oraz ‘rInject’). Jeżeli jest on zerem, to usuwamy ten element z tablicy, gdyż jest to założenie, że w tym cyklu klient nie chce wprowadzić pakietu do medium. Jeżeli jest to natomiast wartość ‘1’ to sprawdzamy, czy dany koniec symulowanego kabla Ethernet nie zawiera w sobie pakietów. Jeżeli zawiera pakiet, to klient oczekuje z wprowadzeniem pakietu do następnego cyklu. Jeżeli natomiast jest pusty, klient wprowadza pakiet do medium oraz usuwany jest pierwszy element tablicy.

## 2.4 Gromadzenie danych

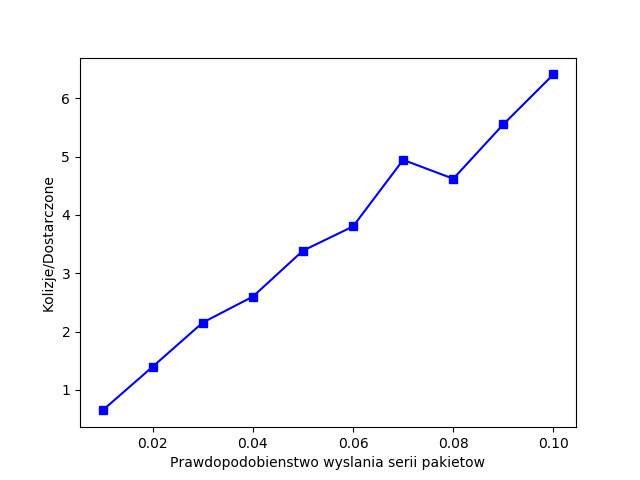
Na potrzeby przeanalizowania czynników mających wpływ na ilość konfliktów oraz ilość dostarczonych pakietów zostały zebrane dwa zestawy danych:

1. Jak zmiana w prawdopodobieństwie wysyłania pakietów wpływa na statystyki?
2. Jak zmiana w długości medium wpływa na statystyki?

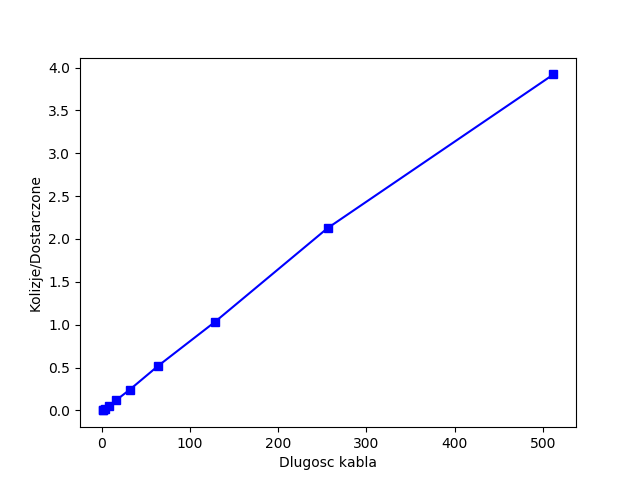
Pierwsza analiza jest szczególnie interesująca, ponieważ przy wyższej liczbie przesyłanych pakietów dużo zależy od czynnika w miarę losowego, którym jest to kiedy dany klient ma zamiar wysłać pakiety. Oczywistym jest, że przy wyższym prawdopodobieństwie przesłania pakietów zarówno ilość dostarczonych pakietów jak i ilość kolizji się zwiększą, dlatego też porównywany zostaje stosunek dostarczonych pakietów do pakietów utraconych.

Druga analiza wydaje się być dosyć oczywistą, przy założeniach w tym zadaniu można zauważyć, że im dłuższe będzie medium tym większa jest szansa, że dwa pakiety z różnych stron zostaną jednocześnie umieszczone w przewodzie. Zgromadzenie danych pozwoli jednak na dokładniejsze przyjrzenie się jak dokładnie zmienia się stosunek utraconych pakietów do ty dostarczonych.

Oba eksperymenty zostały wykonane dla 10 punktów pomiarowych i ilości symulowanych cykli równej 1000. Dodatkowo dla każdego punktu pomiarowego zrobiono 100 powtórzeń a wynik uśredniono. W przypadku pierwszej analizy została założona stała długość kabla równa 100, natomiast w przypadku drugiej analizy zostało założone stałe prawdopodobieństwo wysłania serii pakietów równe 0.01.



Wykres 1: Stosunek kolizji do dostarczonych pakietów a prawdopodobieństwo wysłania serii pakietów



Wykres 2: Stosunek kolizji do dostarczonych pakietów a długość symulowanego medium

## 2.5 Wnioski

Wraz ze wzrostem prawdopodobieństwa wysłania serii pakietów rośnie częstość wysyłania pakietów przez klientów. Oznacza to, że częściej pojawia się szansa na kolizję pomiędzy wysyłającymi stronami. Jednakże wynik uzyskany w przeprowadzonych eksperymentach jak zaskakująco wysoki. Jak widać wraz z liniowym wzrostem prawdopodobieństwa wysłania serii pakietów stosunek kolizji do odebranych pakietów rośnie liniowo. Co jest jednak interesujące to to, że z tak teoretycznie niskim prawdopodobieństwem jak 0.02 ilość kolizji przewyższa już ilość odebranych pakietów. Jednakże myśląc o faktycznej komunikacji poprzez kabel Ethernet i szybkości przesyłu pakietu, nie powinno być to zadziwiające – w porównaniu z prędkością z jaką prąd przemieszcza się w kablu częstość wysyłania pakietów od danego urządzenia jest dosyć niska.

Równie interesujący wynik można zaobserwować w drugim eksperymencie. Oczywiście, wraz ze zwiększeniem długości kabla stosunek ilości kolizji do ilości dostarczonych pakietów się zwiększa, jednakże zwiększanie się tego stosunku jest o wiele mniejsze niż w przypadku zmiany częstości wysyłania pakietów. Jednakże zależność również jest liniowa, co intuicyjnie wydaje się logiczne – z wydłużeniem kabla liniowo rośnie prawdopodobieństwo zderzenia się pakietów.