Technologie sieciowe - sprawozdanie 3

Wojciech Wróblewski

Opis doświadczenia 1

Celem doświadczenia jest stworzenie programu ramkującego zgodnie z zasadą rozpychania bitów oraz weryfikującego poprawnosć ramki metodą CRC. Doświadczenie obejmuje również implementacje procedury odwrotnej w celu uzyskania kopii oryginalnego pliku źródłowego .

Realizacja zadania

- Stworzenie klasy odpowiedzialnej za stworzenie testowego pliku input.txt wraz z jego zawartością w postaci ciągu bitów o zadanej długości. Gdy ciąg zostaje wygenerowany wywołujemy na nim funkcję kodującą. Przez rozpoczęciem kodowania ustalamy parametry charakterystyczne dla protokołu.Dla uproszczenia, ustanawiamy flagę początkową równą końcowej (w postaci bitowej 01111110) . Należy również ustanowić poczatkowy rozmiar podziału danych do ramki o raz wykorzystywany CRC(W doświadczeniu 32 bitowy CRC oraz 32 bitowe ciągi danych).

- Otrzymując jasny podział danego ciągu no podciągi najwyżej 32 bitowe, obliczamy CRC32 dla zadanego ciagu danych oraz stosujemy konkatenację danych z CRC. Nastepnie na zadanym ciągu stosujemy algorytm rozpychania. Algorytm dba o to, żeby w ciągu zadanych bitów nie występowało sześć jedynek pod rząd, dlatego też po każdych pięciu jedynkach następuje rozepchanie ciągu poprzez dodanie '0'. Po wykonaniu procedury dołączamy flagi początkową oraz końcą . Dla uproszczenia w przeprowadzanym doświadczeniu dane flagi mają równą wartość wynoszącą 01111110 .

- Implementacja funkcji dekodującej początkowo jest analogiczna do kodującej. Pobieramy zakodowany ciąg znaków z pliku, dzielimy go ze względu na flagi i poddajemy dekodowaniu. Dzięki flagom początkowej i końcowej wyodrębiamy dane z wnętrza każdej ramki (po podzieleniu ciągu bitów usuwamy flagi początkową i końcową). Na każdej z wyodrębionych porcji danych stosujemy funkcję odwrotną do rozpychania. Teraz oddzielamy dane imformacyjne od CRC gdyż wiemy, że przy stosowaniu CRC32 zajmnie ono ostatnie 32 bity wyodrębionego

kodu z ramki. Teraz mając wyodrębione dane informacyjne (payload) możemy porównać crc32 dla danych z tym który otrzymaliśmy po dekodowaniu .

Określmy prosty protokuł. Maksymalnie 32 bitowy pakiet danych konkatenujemy z CRC32. Na zadanym ciągu przeprowadzamy procedurę rozpychania bitów i nastepnie nadajemy flagi początkową i końcową.


```
Test of stuffing
----------------
data:                               01111111101111111011
stuffed                             011111011101111101011011
success!!!
Test of reverse_stuffing
----------------
data:                               011111011101111101011011
un_stuffed:                         01111111011111111011
success!!!
Test of encoding
----------------
data:                               01111111101111111011
add_flags_(bitstuffing(crc(data)):  01111110011111011101111101011011011100101000010111000001001011101111110
encode(data):                       01111110011111011101111101011011011100101000010111000001001011101111110
success!!!
Test of decoding
----------------
data                                11101
decode(encode(data)) :              11101
success!!!
```

```
given 32 bit sequence:      10001010000011101100001100000111
given 32 sequence encoded:  01111110100010100000111011000011000001110110101010100011100111111011101011011111110
function decode gets:       01111110100010100000111011000011000001110110101010100011100111111011101011011111110
deleted begin&end flag:             10001010000011101100001100000111011010101010001110011111011101011
after reverse_stuffing:             100010100000111011000011000001110110101010100011100111111111011
data:                       10001010000011101100001100000111
unpacked_crc32:             01101010101000111001111111101011
crc32:                      01101010101000111001111111101011
```

Sprawdźmy czy weryfikacja ramki metodą CRC zwróci oczekiwane wyniki.Wprowadźmy jeden błędy bit w ciągu bitów który jest wynikiem funkcji kodującej. Zdekodujmy zadany ciąg. Wyodrębnimy crc z zdekodowanego ciągu jako unpacked _crc oraz obliczmy crc na podstawie zdekodowanych danych (payload). Zauważmy, że program wykrył niezgodnośc kodów CRC.

```
check correctness of crc
-------------------------
encoded string:                01111110100010100000111011000011000001110111110110001000110100111011111010101100101111001111110
encoded string with error bit : 01111110100010100000111011000011000001110111110110001010110100111011111010101100101111001111110
unpacked_crc32:                10110100111011111101011001011110
crc32:                         00110100111011111101011001011110
ERROR.Difference in unpacked crc32 and crc32.
```

Pseudokody implementowanych funkcji.

```
begin_flag = '01111110'
end_flag  = '01111110'
MAX_ONES_LIMIT = 5

procedure stuffing(input)
begin

    stuffed_input.set_empty()
    counter_of_1_bits := 0

    for bit in input
        if bit == '1' do
            counter_of_1_bits += '1'
            stuffed_input   += bit
            if counter_of_1_bits == MAX_ONES_LIMIT do
                   stuffed_input    += '0'
                   counter_of_1_bits == 0
        else
            counter_of_1_bits += 0
            stuffed_input   += bit

    return stuffed_input

end


procedure reverse_stuffing(input)
begin

    unstuffed.set_empty()
    counter_of_1_bits := 0

    for bit in input
          if bit == '1' do
                 counter_of_1_bits += '1'
                 unstuffed += bit
          else
```

```
                if counter_of_1_bits < MAX_ONES_LIMIT
                    unstuffed += bit
                counter_of_1_bits == 0
    return unstuffed

end

procedure add_flags(input)
begin
       return concatinate (begin_flag,input,end_flag)

end


procedure encode(input)

// used on max 32 bit subsequences of input binary sequence
begin

       data = concat(input + crc32(input))
       return add_flags(stuffing(data))
end


procedure decode(input)
//used on smaller subsequences of input binary sequence chuncked
    by split_input_by_flags() method

begin
    CRC_SIZE = 32
    decoded := reverse_stuffing(remove_flags(input))
    data := decoded[0,decoded.length() - CRC_SIZE]
    crc := decoded[decoded.length() - CRC_SIZE, decoded.length()]

    if crc32(data) == crc do
        return data
    else
        return NULL
end
```

Wnioski

Doświadczenie pokazuje w uproszczonym stopniu jak pakowane są informacje podczas przesyłu i pokazuje przykładową strukturę protokołu informacyjnego. Uświadamia, że kontrolowanie metodą CRC jest istotne w efektywnym, sprawnym przekazywaniu informacji, ponieważ daje kontrolę nad występującymi przekłamaniami.

Opis doświadczenia 2

Celem doświadczenia jest stworzenie programu przeprowadzającego symulację ethernetowej metody dostepu do medium transmisyjnego (CSMA/CD).
CSMA/CD - (ang. Carrier Sense Multiple Access with Collision Detection) – protokół wielodostępu CSMA z badaniem stanu kanału i wykrywaniem kolizji.

Opis działania programu

Przyjrzyjmy się strukturze programu przedstawionej w pseudokodzie. Program realizujący protokuł CSMA/CD składa się z klasy Node_Thread, której obiekt jest wątkiem reprezentującym jeden węzeł transmisyjny. W funkcji csma_cd() program pobiera dane o ilości węzłów oraz maksymalnej liczbie pakietów jakie każdy z nich może przesłać, następnie tworzy obiekty klasy Node_Thread, które zaczynają transmisję . Nastepnie program czeka na zakończenie pracy wątków i informuje o zakończenie przesyłu .

```
// main procedure
procedure csma_cd ()
begin
       nodes = get_numer_of_nodes()
       array_nodes_names = set_nodes_names(nodes)
       array_packages = set_pacgages_to_send(nodes)

       for i = 0 to nodes do
           new Node_Thread(name[i] , array_packages[i])

       catch_exception_ on (wait_for_threads_to_join())
       info ( "Transmisja pakietow zakoczona !!!" )

end
```

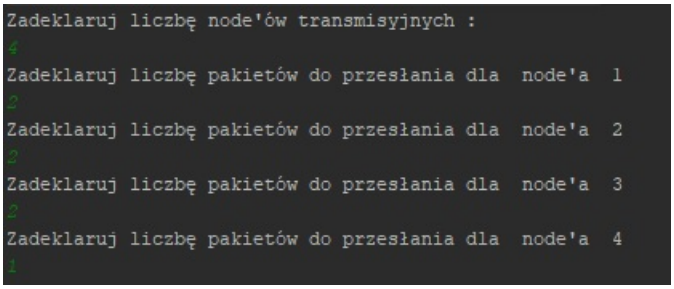
W klasie Node_Thread definiujemy charakterystyczne pola, potrzebne do impementacji algorytmu.W konstrukturze obiektu Node_Thread przypisujemy nazwę oraz maksymalną liczbę pakietów do przesłania i wywołujemy funkcję $run()$, która realizuje transmisję danych zgodnie z protokołem $CSMA/CD$. Definiujemy również zmienną max_number_of_retransmissions, która przechowuje maksymalną liczbę nieudanych prób transmisji pakietu. Jeżeli zostanie ona przekroczona wówczas transmisja zostaje zakończona.

```
// class represents Node objects
class Node_Thread
```

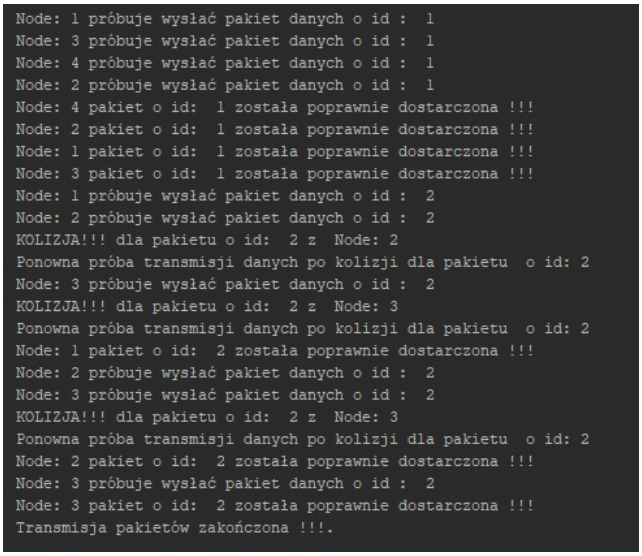
```
begin
        String node_name,channel_status_indicator
        Integer packages[] , distance ,
            max_number_of_retransmissions
            ,current_attempt_to_retransmit

        Node_Thread_constructor (node_name , packages)
                max_number_of_retransmissions.set()
                t = new Thread(node_name)
                t.run()

        prodecure run()
        begin
                while still_has_package_to_deliver do

                    package = get_package(packages)
                    while current_attempt_to_retransmit <
                        max_number_of_retransmissions do

                        if channel_status_indicator == BLOCKED do

                                info ("kana przekazu jest zajety
                                    !!!")
                                thread.sleep(set_random_time())
                            else
                                info ("Node probuje wysac pakiet
                                    danych ")
                                if channel_status_indicator == FREE do

                                    channel_status_indicator ==
                                        BLOCKED
                                    simulate_transmission(package,
                                        distance)
                                    info("Pakiet zostal poprawnie
                                        dostarczony !!!")
                                    channel_status_indicator == FREE
                                else
                                    colission()
                                    info("KOLIZJA!!!")
                                    current_attempt_to_retransmit += 1
                                    thread.sleep(generate_time_to_back_channel())
                                    info("Ponowna proba transmisji
                                        danych po kolizji")
                    end
                     //number of retransmitions exceeded
                         allowable number of retransmissions
                     stop_retransmiting(package)
                     info("Zbyt dua liczba prob dla pakietu
                         .Transmisja pakietu zakonczona.")
```

```
                end
            end
end
```

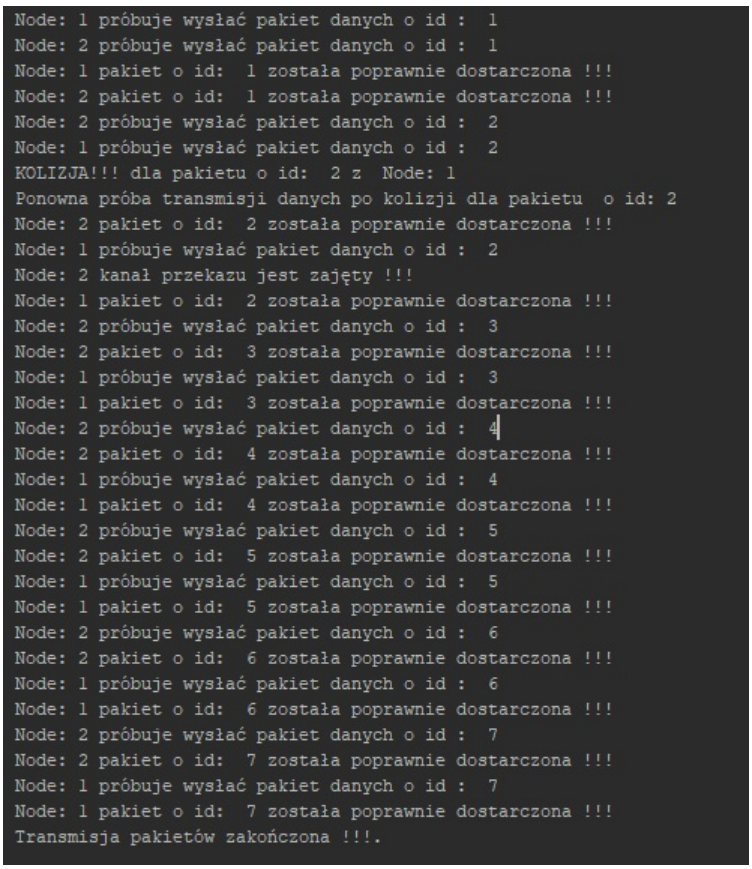
Testy oraz przykładowe zobrazowanie działania funkcji programu.
Program w uproszczonej wersji przedstawia działanie kanału komunikacyjnego . Liczba węzłów komunikacyjnych oraz liczba pakietów, którą każdy węzeł przesyła jest deklarowana podczas działania programu.
Przykładowa deklaracja sieci komunikacyjnej z 4 węzłami . Output programu



przedstawiający symulację CSMA/CD wypisując kolejno stany komunikacji dla zadanego wyżej modelu z 4 węzłami.



Output programu przedstawiający symulację CMSA/CD wypisując kolejno stany komunikacji dla modelu z dwoma węzłami, gdzie każdy ma do przesłania 7 pakietów.



Wnioski

Program symulujący w jasny sposób przedstawia działania protokołu CSMA z wykrywanie kolizji. Analizując wyniki programów widzimy,że protokuł usprawnia przesyłanie danych.Kluczowe zasady protokołu nie tylko wykrywają kolizje, lecz również zmniejszają ryzyko ich wystąpienia, ponieważ ilość wysyłanych pakietów zmniejsza się, a gdy przesył wystapił bezkolizyjnie mamy pewność, że dane zostały przesłane poprawnie. Węzeł jest w stanie wysłać informację tylko wtedy, gdy sieć jest wolna, jednak istnieje możliwość, że dwa węzły nie wiedzą o sobie i wysyłają wiadomość w tym samym momencie. Protokuł dobrze radzi sobie z takimi zdarzeniami, wówczas zgłasza kolizję i naliczając pewne opóźnienia kolejkując nam wysyłanie informacji i zwiększając niezawodność przesyłu.