Filip Kulasza

*Technologie Sieciowe*

Sprawozdanie - lista 3

1. **Ramkowanie**

Zadanie polega na ‘ramkowaniu’ ciągów bitów (znaków) z pliku, a następnie zapisaniu ich do nowego. Stworzony przeze mnie program posiada 2 tryby:

* Code – kodujący (ramkujący) ciąg bitów z pliku o zadanej ścieżce do pliku o zadanej ścieżce
* Decode – odkodowujący ciągi będące ramkami (wykrywa, gdy ramka jest nieprawidłowo stworzona)

Dane ramkowane dzielone są na określoną ilość bitów na pakiet.

Przygotowałem dane testowe (plik „test.txt), które najpierw za pomocą programu ‘ramkuję’, poprzez poddanie działaniu procedurze „encode” a następnie wykonuję operację odwrotną. Dane ramkowane są w pakiety po 32 bitów za pomocą 8-bitowego CRC z podzielnikiem „100110001”.

Wejściowy ciąg bitów:

11110101111011111101011101111111000110001010100111011110101

Po ramkowaniu:

01111110111101011110111110101011101111101111100011101111110011111100001100010101001110111101010010111001111110

Po operacji odwrotnej:

11110101111011111101011101111111000110001010100111011110101

Operacja przebiegła zatem pomyślnie, a z kodu po ramkowaniu bez większej trudności możemy wydzielić dopisane ciągi znaków na końcu i początku pakietu oraz kody crc.

1. **Symulacja dostępu do medium transmisyjnego**

W sieci Ethernet urządzenia korzystają ze wspólnego medium transmisyjnego, co może prowadzić do kolizji spowodowanej próbą nadawania przez różne urządzenia w tej samej chwili czasu. Protokół CSMA/CD ogranicza powstawanie kolizji w medium transmisyjnych i opisuje zasady postępowania w wypadku wykrycia kolizji. Procedura nadawania w tym protokole przebiega następująco: Przed rozpoczęciem nadawania urządzenie sprawdza, czy medium transmisyjne jest wolne. Jeśli tak, rozpoczyna nadawanie i kontynuuje je przez czas dwukrotnie dłuższy niż czas konieczny do dotarcia wiadomości do końca medium transmisyjnego. Przez cały okres nadawania urządzenie sprawdza też, czy nie nastąpiła kolizja w transmisji. Jeśli tak się stało, urządzenie kontynuuje nadawanie, przy czym nagłówek wiadomości zastępuje sygnałem kolizji (jam). Zwiększany jest także licznik prób. Jeśli przekroczona została maksymalna liczba prób wysłania pakietu, transmisja zostaje przerwana. Jeśli nie, urządzenie losuje (z zakresu 2liczba prób − 1), ile jednostek czasu będzie czekać zanim wznowi transmisję.

* 1. **Realizacja**

W celu realizacji tej części ćwiczenia również stworzono program w języku Java. Program składa się z klas: Host reprezentującej urządzenie nadawcze, Wire reprezentującej medium transmisyjne, WireCell reprezentującej jedną „komórkę” (jednostkę długości) tego medium oraz głównej klasy EthernetSimulator zarządzającej całą siecią.

* + 1. **Klasa Host**

Klasa Host odpowiada za wysyłanie kolejnych pakietów wiadomości, wykrywanie kolizji i reagowanie na nie. Główna funkcjonalność tej klasy umieszczona została w metodzie send() wywoływanej za każdym przebiegiem głównej pętli programu. Klasa ta zawiera także flagi sending, jam, collision, wantToSend (odpowiadające odpowiednio za różne stany nadawania: wysyłanie, wysyłanie sygnału błędu, wykryto kolizję, oczekiwanie na możliwość wysłania pakietów). Metoda send() sprawdza wartości tych flag i w zależności od nich podejmuje odpowiednie działania. Jeśli host jest w stanie jam i ma jeszcze pakiety do wysłania, wysyła je do swojej wireCell i dekrementuje licznik pakietów do wysłania. Jeżeli host jest w stanie kolizji, ale nie wysyła już informacji o błędzie, oblicza czas oczekiwania przed następną próbą nadawania (calculateBackoff()) i inkrementuje licznik prób ponownego wysłania pakietów. Jeżeli licznik przekroczył maksymalną dopuszczalną wartość (ustaloną na 10), kończy nadawanie. Jeżeli host jest w stanie wysyłania i ma nadal pakiety do wysłania, wysyła je do medium transmisyjnego i sprawdza, czy nie nastąpiła kolizja. Jeśli nastąpiła, ustawia flagi jam, collision i wantToSend. Jeżeli host nie jest w stanie nadawania, ale chce nadawać (wantToSend) i nie oczekuje na następną próbę nadania, a medium transmisyjne jest wolne, host przechodzi w stan nadawania.

* + 1. **Klasa WireCell**

Klasy WireCell przechowuje wszystkie sygnały, które w danym momencie znajdują się w danej komórce medium transmisyjnego. Metoda getState() tej klasy zwraca stan danej komórki pamięci (jam, collision, empty lub znajdującą się w niej wiadomóść) w zależności od tego ile i jakich sygnałów znajduje się aktualnie w tej komórce medium.

* + 1. **Klasa Wire**

Głównym zadaniem klasy Wire jest propagacja pakietów w medium transmisyjnym. Przy każdym przebiegu głównej pętli programu wywoływana jest metoda propagate(), która przechodząc przez wszystkie komórki medium przesuwa sygnały poruszające się w lewo o jedną komórkę w lewo, sygnały poruszające się w prawo o jedną komórkę w prawo, a sygnały poruszające się w obu kierunkach (w punktach, w których są one nadawane) i w prawo, i w lewo.

* + 1. **Klasa EthernetSimulator – główna pętla programu**

Metoda run() klasy EthernetSimulator zawiera główną pętlę programu, która kolejno: wywołuje metodę send() dla wszystkich hostów obecnych w sieci, wypisuje na ekran aktualny stan medium transmisyjnego, wywołuje metodę propagate() na medium.