Wrocław, 10.05.2017

Laboratorium

Technologii Sieciowych

Temat:

Sprawozdanie 3

Autor: Bartosz Banasik
Informatyka, semestr: 4
Prowadzący: dr Łukasz Krzywiecki

1. Ramkowanie metodą rozpychania bitów

1.1 Cele pracy

Celem pierwszej cześci zadania było napisać program ramkujący zgodnie z zasadą „rozpychania bitów”, oraz sprawdzenie poprawności ramki za pomocą metody CRC. Program ma odczytywać plik źródłowy ‘Z.txt’ zawierający dowolny ciąg złożony ze znaków `‘0’ i ‘1’` (symulujący strumień bitów) i zapisywać ramki odpowienio sformatowane do pliku tekstowego ‘W.txt’.

Napisać program realizujący procedurę odwrotną, tzn. który odczytuje plik ‘W.txt’ i dla poprawnych danych CRC przepisuje jego zawartość tak, aby otrzymać kopię oryginalnego pliku źródłowego.

1.2 Realizacja

Program został napisany w języku `JAVA.`

Główna metoda programu sprawdza czy wybraliśmy opcję `code_and_write,` czy opcję `decode_and_write.` W przypadku pierwszej opcji, program wczytuje plik i poddaje go kodowaniu. Cały strumień bitów dzielony jest na porcje po 32 bity. Dla każdej porcji obliczany jest 8 bitowy kod CRC, który następnie jest dodawany do bierzącej ramki. Całość teraz poddawana jest „rozpychaniu bitów”. Polega to na tym, że jeśli napotkamy pięć jedynek pod rząd, to po ostatniej z nich dodajemy 0. W ostatnim kroku dodajemy flagi początku i końca ramki (`„01111110”`). Całość zapisywana jest do pliku W.txt.

W przypadku wyboru opcji drugiej, program rozdziela ramki na osobne porcje znaków. Następnie z każdej usuwane są flagi początku i końca. W następnym kroku program usuwa niepotrzebne 0 dodane przy kodowaniu. Teraz pozostało nam już tylko sprawdzić kod CRC czyli ostatnie 8 bitów ramki. Jeśli kod CRC się zgadza to zapisujemy odkodowany ciąg do pliku.

1.3 Kod programu odpowiedzialny za ramkowanie

```
129     private static void code_and_write(String data) {
130 
131         //otwieramy strumień do zapisu
132         BufferedWriter out = null;
133         try {
134             out = new BufferedWriter(new FileWriter("./W.txt"));
135         } catch (IOException e) {
136             e.printStackTrace();
137         }
138 
139         String frame;
140         int m = 32;
141         try {
142             //Dzielimy strumien bitów na porcje po 32 bity
143             for (int n = 0; n < data.length(); n += m) {
144                 //jesli nie ma 32 to bierzemy to co zostało
145                 if (n + m > data.length()) {
146                     m = data.length() - n;
147                 }
148                 frame = data.substring(n, n + m);
149                 /*
150                  * Obliczamy pole kontrolne CRC
151                  */
152                 String crc = checkSum_CRC8(frame);
153                 frame += crc;
154                 /*
155                  * Stosujemy metody bite stuffing
156                  */
157                 frame = biteStuffing(frame);
158                 /*
159                  * Dodajemy flagi poczatku i konca
160                  */
161                 frame = "01111110" + frame + "01111110";
162 
163                 out.write(frame);
164                 out.write("\n");
165 
166             }
167             out.close();
168         } catch (IOException e) {
169             e.printStackTrace();
170         }
171     }
```

Text 1: Listing zawierajacy kodowanie ramki

1.3 Przykladowe wywolanie programu

W pliku ‘Z’ zapiszemy ciąg:

```
0000111111000111011101111010101011110101010101010101010101010111
```

Text 2: Zawartosc pliku 'Z'

Uruchamiamy program z opcja kodowania.

Poniższy listing zawierazawartość pliku ‘W’:

```
1 01111110000011111010001110111011110101010111101000111111 0
2 01111110111101010101010101010101010101011101010010011111 10
```

Text 3: Zawartość pliku 'W'

Jak widzimy bardzo łatwo można zauważyć flagi początku oraz końca. Po bliższym spojrzeniu na sekwencje widzimy, że po kązdym ciągu pięciu jedynek nastepuje 0.

Uruchamiamy program w opcji decode dla pliku ‘W’ wymienionego powyżej. Poniżej znajduje się trść pliku ‘Z_copy’vim:

```
1 0000111111000111011101111010101
2 1111010101010101010101010101010111
```

Text 4: Zawartość pliku 'Z_copy'

1.4 Analiza wyników

Program wykonał prawidłowo wszystkie zadania. Poprawnie zakodował dane wg podanej metody. Metoda ramkowania określa w sposób jednoznaczny prawidłowość przesłanych danych przez co ma duże zastosowanie praktyczne w sieciach komputerowych.

2. Symulacja metody dostępu do miedium transmisyjnego

2.1 Cel

Napisać program (grupę programów) symulujacych ethernetow a ˛ metod ˛e dostepu do medium trans-misyjnego (CSMA/CD).

2.2 Realizacja

Program został napisany w języku `JAVA` i składa się z 3 klas `Network.java, Host.java, Signal.java.`

Zasada działania `Network.java`

Klasa tworzy tablice imitujaca zachowanie kabla. Inicjalizuje dla niej 3 Hosty.

Natępnie wykonuje 100 iteracji symulujących propagacje sygnału w kablu transmisyjnym. Sprawdza czy któreś sygnały się pokrywają. Nanosi poprawki dla naszego kabla. Sprawdza czy dojdzie do kolizji sygnałów. Dla każdego hosta losuje prawdopodobieństo, czy chce on nadawać wiadomość, oraz wykonuje przemieszczenie sygnałów.

```
62    private static void loop()
63     {
64        // Likwidacja sygnalow ktore juz skonczyly
65        for(int i=0; i<signals.size(); i++)
66        {
67           if(signals.get(i).getSignalDuration() <= 0)
68              signals.remove(i);
69        }
70
71        // Tablica przechowujaca nalozone sygnaly
72        List<String> array = new ArrayList<>();
73        for(int j=0; j<length; j++) array.add("");
74
75        char error = '*';
76        for (Signal signal : signals) {
77           int position = signal.getPosition();
78           String message = Character.toString(error);
79           if (signal.getMaster() != null)
80              message = Character.toString(signal.getMaster().getMessage());
81
82           if (!array.get(position).contains(message)) {
83              String current = array.get(position);
84              String changed = current + message;
85              array.set(position, changed);
86           }
87        }
88
89        // Nanoszenie poprawek na kabel i drukowanie
90        for(int i=0; i<cable.length; i++)
91        {
92           if(array.get(i).length() == 0)
93              cable[i] = empty;
94           else if(array.get(i).length() == 1)
95              cable[i] = array.get(i).charAt(0);
96           else
97             cable[i] = error;
98
99           System.out.print(cable[i]);
100       }
```

```
101
102         //Sprawdzanie czy dojdzie do kolizji sygnałów. Jeżeli tak to tworzy się  zakłócenie
103         for(int i=0; i<signals.size(); i++)
104         {
105             for (Signal signal : signals) {
106                 Signal p = signals.get(i);
107
108                 //PQ lub P_Q
109                 if (p.getPosition() + 1 == signal.getPosition() || p.getPosition() + 2 == signal.getPosition()) {
110                     if (p.getDirection() && !signal.getDirection()) {
111                         if (p.getMaster() != signal.getMaster()) {
112                             signals.get(i).lose();
113                             signal.lose();
114                         }
115                     }
116                 }
117             }
118         }
119
120         // Losowanie hostów do wysłania wiadomości
121         for (Host host1 : hosts) {
122             double probability = new Random().nextDouble();
123             double p = 0.01;
124             if (probability < p) {
125                 host1.start();
126             }
127         }
128
129         //Hosty - rozpatrzenie przypadków
130         for (Host host : hosts) {
131             int position = host.getPosition();
132             char message = host.getMessage();
133
134             if (host.isWaiting()) {
135                 host.waitForTransmission();
136             } else {
137                 if (host.wantToTransmit()) {
138                     if (cable[position] == empty || cable[position] == message) {
139                         signals.add(host.createSignal(true));
140                         signals.add(host.createSignal(false));
141                     } else {
142                         host.error();
143                     }
144                 }
145             }
146         }
147
148         //Przesuwanie sygnałów
149         for (Signal signal : signals) {
150             signal.move();
151         }
152     }
```

Text 5: Funkcja imitująca propagacje sygnału

2.3 Wnioski

Program symuluje działanie medium transmisyjnego CSMA/CD. Przesył danych odbywa się w momencie kiedy host nie zauważa aktywności nad swoim źródłem. Jednak w przypadku kiedy dochodzi do kolizji, tzn. co najmniej dwa urządzenia nadają w tym samym czasie do tego samego medium transmisyjnego, przesył jest zatrzymany, a program wysyła komunikat o wystąpieniu kolizji. Program symulujący w jasny sposób pokazuje działanie medium CSMA/CD z wykrywaniem kolizji. Metoda sprawia wrażenie sprawnej i przez to jest bardzo szeroko stosowana w sieciach komputerowych.