Laboratoria z Technologii Sieciowych

Lista 3

Damian Kokot

**1. Zadanie 1**

Celem tego zadania było napisanie programu, który ramkuje podaną wiadomość metodą rozpychania bitów. W ramce powinno znaleźć się pole CRC, odpowiedzialne weryfikację poprawności otrzymanej ramki. Dzięki programowi można również dokonać odwrotnej procedury, czyli odczytać plik wynikowy, zweryfikować poprawność ramki metodą CRC i zwrócić wiadomość jeżeli ramka była poprawna.

*Zadanie zrealizowane jest w technologii* ***NodeJs****.*

1. W pierwszej kolejności odczytujemy plik tekstowy z.txt z wiadomością do obramkowania.
2. Potem dzielimy na ramki o pewnych stałych długościach (np po 100 bitów)
3. Następnie obliczamy CRC32. Robimy to za pomocą biblioteki 'buffer-crc32'. Obliczone CRC jest ciągiem 0-1 o długości 32.
4. Kolejnym krokiem jest rozszerzanie wiadomości wykrywając każdą sekwencję pięciu jedynek i wpychając zero na końcu każdej z tych sekwencji.
5. Ostatnim krokiem jest nadanie sygnałowi znaku początku i końca. Dodajemy zatem przed i po powyższej sekwencji ciąg 01111110

*Plik z.txt*

111111111111111111110100011111111101111111101 // Obliczamy CRC:

wiadomość CRC 111111111111111111110100011111111101111111101 11011101101100000101100100100110

// Po rozepchaniu bitów:

111110111110111110111110010001111101111011111011101 11011101101100000101100100100110

Jak widzimy dla każdego z sześciu przypadków rozszerzyliśmy ciąg pięciu jedynek o zero otrzymując w wyniku ciąg o sześć elementów dłuższy. Istotnym jest aby uruchomić powyższą operację po obliczeniu CRC.

*Plik z.txt*

11111111111111111111

$ node main.js -e

01111110 1111101111101111101111100 11011101101100000101100100100110 01111110

| Sign | Message | CRC | Sign |

Bardzo analogicznie postępujemy w drugą stronę. Umownie traktujemy źródło danych jako plik tekstowy a nie jako łącze.

1. Wyszukujemy znaku początku i końca wiadomości ‘01111110’, po czym odramkowywyjemy wiadomość po wiadomości.
2. Dla każdej wiadomości najpierw pozbywamy się znaku początku i końca.
3. Następnie pozbywamy się sekwencji 111110 na rzecz 11111.
4. Kolejnie obliczamy CRC wiadomości i porównujemy z otrzymanym CRC z ramki. Jeżeli choć jeden bit jest zmieniony, ramka okaże się być błędna, a program wyrzuca nam błąd:  
   Error: Data CRC doesn't match
5. Jeżeli nie ukończyliśmy wczytywania wiadomości, wyszukujemy najbliższy znak początku ignorując znaki przed nim, po czym przechodzimy do kroku

**2. Zadanie 1**

Celem zadania drugiego jest napisanie programu symulującego ethernetowej metody dostępu do medium transmisyjnego (CSMA/CD).

*Program napisany jest w technologii* ***NodeJs****.*

Stworzono cztery pliki symulujące sieć ethernet.

W pliku ***main.js*** specyfikujemy sieć podając w argumentach długość łącza i ilość iteracji, przez które sieć ma być aktywna. Następnie dodajemy stacje podając identyfikator, pozycję na łączu i prawdo-podobieństwo wysyłania wiadomości gdy łącze jest wolne (nie jest zajęte), po czym uruchamiamy sieć.

function main() {

const web = new Ethernet(70, 2000)

const interval = 100;

web.attachStation(new Station('A', 5, 0.1)); web.attachStation(new Station('B', 35, 0.1)); web.attachStation(new Station('C', 65, 0.1));

web.run(interval);

}

Kolejnym plikiem jest plik ***station.js***, w którym specyfikujemy klasę Station. Klasa Station reprezentuje stację podłączoną do sieci. Można mamy tu m. in. metody umożliwiające wysył danych, i metody pozwalające na śledzenie obecnego stanu łącza.

Całość jest odwzorowana jest w krokach (steps), mające odzwierciedlić najmniejszą możliwą podzielność czasu. Dla każdej stacji zawarłem metodę *nextStep()*, która jest uruchamiana w każdej iteracji. Zasada jej działania jest następująca:

1. Sprawdzamy, czy na nad stacją znajduje się sygnał konfliktu (umownie '!'). Jeżeli tak, to rozsyłamy sygnał zagłuszający (Jamming signal - 'J', który ma mieć długość 2 \* długość łącza).
2. Jeżeli aktualnie wysyłaliśmy jakiś komunikat, musimy odznaczyć, że nastąpił konflikt przy wysyłaniu wiadomości. Następnie jeżeli ilość konfliktów przy wysyłaniu jest w zakresie :
   * 0 - 10, to obliczamy czas oczekiwania, który jest równy:

*2 \* długośćŁącza \* losowa liczba z przedziału 0 - 2 ^ ilośćKonfliktów*.

* + 10 - 16, to obliczamy czas oczekiwania, który jest równy *2 \* długośćŁącza \* 2 ^ 10*.
  + jeżeli oczekiwanie na łącze trwa dłużej, wyrzucamy błąd.

Oczekiwanie rozpoczynamy po zakończeniu wysyłania sygnału JAM.

1. Jeżeli nie wysyłamy żadnej wiadomości, losujemy prawdopodobieństwo wysłania i jeżeli jest ono mniejsze niż prawdopodobieństwo podane na starcie, uruchamiamy.
2. Jeżeli jesteśmy w trakcie wykonywania wysyłania wiadomości / oczekiwania - kontynuujemy.
3. Jeżeli udało nam się wysłać pomyślnie pakiet, zerujemy liczbę konfliktów.

Plik *ethernet.js* służy jako łącze pomiędzy przewodem, a stacją. Udziela informacji o stanie łącza, obsługuje sygnały wysyłane przez stację i wypisuje każdy krok działania programu.

Plik *wire.js* obrazuje łącze. Używa ona klasy Signal do zarządzania sygnałem, obrazuje falę (lewą i prawą) sygnału, zarządza poruszaniem się sygnału i odświeżaniem stanu sygnałów.

class Wire {

broadcastSignal(signal) {

this.signals.push(new Signal(signal, this.wireLength));

}

updateSignal() {

* Odświeżamy sygnał. Jeżeli sygnał miał falę w punkcie 'i' i 'j', sygnał powinien się przenieść na punkt 'i - 1' i 'j + 1';

this.signals.forEach(signal => signal.updateSignal());

* Usuwa sygnały, które wyszły poza zakres tablicy (skończyły się) this.signals = this.signals.filter(signal => {

return signal.leftPos !== null || signal.rightPos !== null;

});

}

updateStates() {

this.states.fill(Signal.none);

// Dla każdego sygnału

for (let signal of this.signals) {

// Dla pozycji lewej i prawej

for (let position of [signal.leftPos, signal.rightPos]) { if (position !== null) {

* Zaktualizuj sygnał. this.states[position] = getSignalValue(

this.states[position],

signal.value,

)

}

}

}

}

}

**2.1.** Przykładowe uruchomienieprogramu i analiza:

Stacja A i B nadaje

[AAAAAAAAAAAAAAAAAAA BBBBBBBBBBBBBBBBBBBBB ]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [ | | |  | | | | | ] |
| [ | A |  | B | C | ] |
|  | State | Left: | Colisions in row: | Waiting: | |
| A: | packet | 126 | 0 | 0 |  |
| B: | packet | 129 | 0 | 0 |  |
| C: | none | 0 | 0 | 0 |  |

Iteration: 16/2000

Dochodzi do konfliktu:

[AAAAAAAAAAAAAAAAAAAAA!!BBBBBBBBBBBBBBBBBBBBBBBBBBB ]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [ | | |  | | | | | ] |
| [ | A |  | B | C | ] |
|  | State | Left: | Colisions in row: | Waiting: | |
| A: | packet | 122 | 0 | 0 |  |
| B: | packet | 125 | 0 | 0 |  |
| C: | none | 0 | 0 | 0 |  |

Iteration: 20/2000

Stacja B zauważyła konflikt i rozpoczęła nadawanie synału zagłuszającego 'J'

[AAAAAAA!!!!!!!!!!!!!!!!!!!!!!!!!!!!J!BBBBBBBBBBBBBBBBBBBBBBBBBBB ]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [ | | |  | w | | | ] |
| [ | A |  | B | C | ] |
|  | State | Left: | Colisions in row: | Waiting: | |
| A: | packet | 108 | 0 | 0 |  |
| B: | jamming | 139 | 1 | 140 |  |
| C: | none | 0 | 0 | 0 |  |

Iteration: 34/2000

Stacja A Również zauważa konflikt i uruchamia wysyłanie syngału jam.

[AAAA!J!!!!!!!!!!!!!!!!!!!!!!!!!!JJJJJJJ!BBBBBBBBBBBBBBBBBBBBBBBBBBB ]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [ | w |  | w | | | ] |
| [ | A |  | B | C | ] |
|  | State | Left: | Colisions in row: | Waiting: | |
| A: | jamming | 139 | 1 | 0 |  |
| B: | jamming | 136 | 1 | 140 |  |
| C: | none | 0 | 0 | 0 |  |

Iteration: 37/2000

Zauważmy że obie stacje teraz oczekują na koniec sygnału Jam, po czym stacja A wylosowała wstrzymanie nadawania przez 140 rund.

[JJJJJJJJJJJJJJJJJJJJJJJJJJJJJ ]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [ | w |  | | | | | ] |
| [ | A |  | B | C | ] |
|  | State | Left: | Colisions in row: | Waiting: | |
| A: | none | 0 | 1 | 75 |  |
| B: | none | 0 | 1 | 0 |  |
| C: | none | 0 | 0 | 0 |  |

Iteration: 241/2000

Następnie stacja B zaczęła nadawać i wysyłać sygnał 'B', który ma trwać 140 rund.

[ BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB ]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [ | w |  | | | | | ] |
| [ | A |  | B | C | ] |
|  | State | Left: | Colisions in row: | Waiting: | |
| A: | none | 0 | 1 | 23 |  |
| B: | packet | 106 | 1 | 0 |  |
| C: | none | 0 | 0 | 0 |  |

Iteration: 293/2000

Przy zakończeniu wysyłania sygnał jeszcze przez chwilę będzie się roznosił. NIe musi trwać długo, aby stacja zaczęła znów nadawać.

[BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [ | | |  | | | | | ] |
| [ | A |  | B | C | ] |
|  | State | Left: | Colisions in row: | Waiting: | |
| A: | none | 0 | 1 | 0 |  |
| B: | packet | 140 | 0 | 0 |  |
| C: | none | 0 | 0 | 0 |  |

Iteration: 401/2000

Niestety A zaczęło nadawać w przerwie i doszło do konfliktu.

[JJJJJJJJJJJ!BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [ | w |  | | | | | ] |
| [ | A |  | B | C | ] |
|  | State | Left: | Colisions in row: | Waiting: | |
| A: | jamming | 134 | 2 | 280 |  |
| B: | packet | 102 | 0 | 0 |  |
| C: | none | 0 | 0 | 0 |  |

Iteration: 439/2000

**3. Wnioski**

W zadaniu pierwszym widzimy jak łatwo można uzyskać pewność, że pakiet dotarł w pożądanej postaci. Jesł bardzo male prawdopodobieństwo, że CRC będzie miało taką samą wartość dla błędu w pakiecie, jednak jest ono tak małe, że pozwala na spokojny przesył danym pomiędzy serwerem, a klientem.

W zadaniu drugim, możemy przekonać się o tym jak problematycznym jest zsynchronizowanie łącza internetowego. Jest to o tyle trudne, gdyż trudno jest sprawić, aby tylko jedna stacja miała prawo głosu w danej chwili, skoro nie mamy natychmiastowej informacji o sygnałach nadawanych przez inne stacje.

Przy dużej długości łącza lub dużej ilości stacji może się zdarzyć, że zostaniemy zagłuszeni i za każdym razem, gdy pragniemy coś powiedzieć, wywołujemy błąd. Jest to bardzo problematyczne.