Sprawozdanie z Listy 3 Technologie Sieciowe Mateusz Gancarz

Zadanie 1 - Ramkowanie

W pierwszym zadaniu musieliśmy napisać program, który umożliwi nam ramkowanie danych zgodnie z zasadą rozpychania bitów podaną na wykładzie. Takie ramki składają się z:

-flag informujących o początku i końcu wiadomości (ciąg bitów “01111110”),

-wiadomości,

-wartości kontrolnej CRC (wykorzystamy do tego bibliotekę crc32 dostępną w pythonie).

```
def main():
    with open("test.txt", 'r') as dataFile:
        with open("encoded.txt", 'w') as answerFile:
            answerFile.write(encode(dataFile.read()))

    with open("encoded.txt", 'r') as dataFile:
        with open("decoded.txt", 'w') as answerFile:
            answerFile.write(decode(dataFile.read()))
```

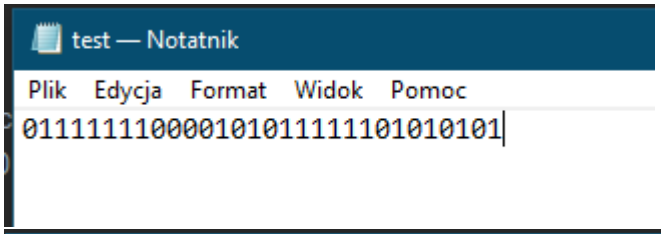
Nasze dane będziemy brać z pliku tekstowego “test.txt” i zapisywać zakodowane dane do pliku tekstowego “encoded.txt”, a następnie odkodowane dane zapiszemy do pliku “decoded.txt”.

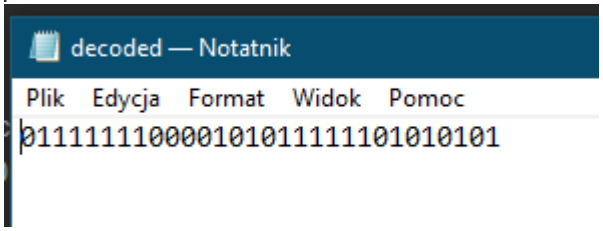
```
def encode(binary_code):
    crc = "{0:b}".format(crc32(binary_code.encode()))
    binary_code += "0" * (CRC_SIZE - len(crc)) + crc

    answer = ""
    counter = 0

    for bit in binary_code:
        answer += bit
        if bit == "1":
            counter += 1
            if counter == 5:
                answer += "0"
                counter = 0

        else:
            counter = 0

    return FLAG + answer + FLAG
```

Do kodowania używamy funkcji encode. Za pomocą biblioteki crc32 generujemy kod crc i umieszczamy go na końcu wiadomości (po dopełnienu długości do 32 bitów dodając “0”). Następnie szukamy sekwencji 5 jedynek i dodajemy po nich “0”, aby nie tworzyły się fałszywe flagi. Na koniec dodajemy flagę “01111110” przed i po wiadomości.

```python
def checkCRC(code):
    crc = "{0:b}".format(crc32(code[:-CRC_SIZE].encode()))
    crc = "0" * (CRC_SIZE - len(crc)) + crc
    return crc == code[-CRC_SIZE:]

def decode(binary_code):
    code = ""

    for suite in binary_code.split(FLAG):
        if suite != "":
            answer = ""
            counter = 0

            for bit in suite:
                if bit == "1":
                    answer += bit
                    counter += 1
                else:
                    if counter < 5:
                        answer += bit
                    counter = 0

            if checkCRC(answer):
                code += answer[:-CRC_SIZE]
            else:
                print("Wrong crc code")
                exit(-1)

    return code
```

Do dekodowania użyjemy funkcji decode. Na samym początku usuwamy flagi “01111110” z zakodowanej wiadomości. Następnie usuwamy “0” występujące po sekwencjach 5 jedynek. Na końcu sprawdzamy poprawność kodu crc i usuwamy go.





Jak widzimy po testach, nasz program poprawnie zakodował i odkodował dane widoczne wyżej.

Zadanie 2 – symulacja CSMA/CD

W protokole CSMA/CD urządzenie rozpoczyna transmisję, gdy wykryje, że łącze jest wolne. Jeśli dwa lub więcej urządzeń rozpocznie transmisje w podobnym momencie, takie zdarzenie spowoduje kolizję i jeśli urządzenie to odkryje, uruchamia procedurę rozwiązywania konfliktu. Przestaje transmitować dane i zamiast tego wysyła jam signal, dzięki któremu inne urządzenia wiedzą, że wystąpiła jakaś kolizja i mogą odrzucić błędną wiadomość. Następnie wybiera czas oczekiwania ze zbioru {0, …, 2^n}, gdzie n = min(k, 10), a k jest liczbą kolizji, które wystąpiły podczas prób transmisji. Po odczekaniu urządzenie podejmuje kolejne próby transmisji.

```
7   int main() {
8       int wireLength = 0;
9       int devicesAmount = 0;
10      float probability = 0.0;
11      int roundsAmount = 0;
12
13      //TEST
14      std::vector<int> places;
15      places.push_back(0);
16      places.push_back(60);
17
18      wireLength = 80;
19      devicesAmount = 2;
20      probability = 0.3;
21      roundsAmount = 500;
22
23      Ethernet ethernet(devicesAmount, places, wireLength, probability, roundsAmount);
24      ethernet.run();
25
26      return 0;
27  }
```

Aby uruchomić program, ustalamy liczbę urządzeń, ich lokalizację, długość węzła, prawdopodobieństwo transmisji urządzenia oraz liczbę iteracji programu. Wiadomość wysłana przez urządzenie 1 jest oznaczona jako ”1” i analogicznie dla urządzenia 2. Zakłócona wiadomość jest oznaczona symbolem ”=”, a jam signal symbolem ”X”.