Lista 4

Zadanie zwiazane z Lista nr 4 polega glownie na takim wykorzystaniu potwierdzen I numerow sekwencyjnych przez nadawce I odbiorce, aby odbiorca wydrukowal wszystkie pakiety w kolejnosci ich numerow sekwencyjnych. Symulacja transmisji pakietow za pomoca protokolu TCP, pomiedzy nadawca a odbiorca odbywa sie przez klase Z2Forwarder, ktora symuluje opoznienia, zagubienie pakietu a takze duplikacje. Komunikacja miedzy nadawca a odbiorca odbywa sie przy pomocy z gory ustalonej szerokosci okna tzn. Ilosci pakietow przesylanych jeden po drugim bez koniecznosci potwierdzenia odbioru przez odbiorcy. Nadawca wysyla pakiety do odbiorcy I czeka okreslona ilosc czasu na powtwiedzenie odboiru (czas ten nazywa sie Round Tip Time – RTT), jesli w tym czasie nie odbierze zadnych informacji od odbiorcy uznaje, ze odbiorca nie otrzymal pakietow I retransmituje ponownie ta sama liczbe paczek z ta sama zawartoscia. W przeciwnym wypadku, to znaczy, jesli nadawca otrzyma potwierdzenie to w potwierdzeniu znajduje sie numer sekwencyjny ostatniego pakietu ktory tworzy spojny ciag odebranych pakietow I na podstawie tych informacji przesyla kolejna porcje pakietow (jesli odbiorca bedzie mial niekompletna sekwencje odebranych pakietow, to po uplywie okreslonego czasu to znaczy takiego w ktorym nalezy juz przeslac potwierdzenie do nadawcy, zapomina o tych pakietach I zachwouje tylko te ktore tworza spojny ciag). W tym momencie warto przyjrzec sie jak to wyglada od strony inplementacji. W pierwszej kolejnosci chcialbym omowic stale parametry programu.

```
static final int senderWindow=10;
static final int RTT = 3000;
static final int datagramSize=50;
static final int packetNumber = 45;
static final int sleepTime=500;
static final int maxPacket=50;
private volatile boolean receivedConfirmation = false;
private volatile int currentByteIndex = 0;
```

Stala senderWindow okresla liczbe pakietow przesylanych w jednym “oknie”, stala RTT okresla wczesniej juz omowiony czas oczekowania nadawcy na potwierdzenie otrzymania pakietow ze strony odbiorcy, stala packetNumber okresla liczbe pakietow ktora zostanie wyslana (jej wartosc zalezy od wielkosci pliku wejsciowego), zmienna sleepTime okresla przerwe miedzy kolejno wysylanymi pakietami (oczywiscie jesli uruchamiamy symulacje przez Z2Forwarder to dochodzi jeszcze opoznienie pakietu), zmienna receivedConfiramtion odpowiada za to zeby poinformowac watek wysylajacy pakiety, ze otrzymano potwierdzenie jesli watek ten jest w stanie oczekiwania, zmienna currentIndex mowi nam jaka literke aktualnie odbiorca odebral.

Wazna czescia programu jest niewatpliwie klasa odpowiedzialna za wysylanie pakietow. Klasa ta na poczatku pobiera I zapisuje zawartosc argumentow wejsciowych to listy o nazwie input, nastepnie wpadamy w petle while ktora wykonuje sie tak dlugo, az nie bedziemy mieli pewnosci. Ze odbiorca nie odebral calej wiadomosci. Pozniej wpadamy w petle for ktora wysyla tyle pakietow ile wynosi wielkosc okna, wyczekujac okreslana ilosc czasu pomiedzy kazdym pakietem. Po przeslaniu wszystkich pakietow oczekujemy odpowiedzi ze strony odbiorcy, jesli taka otrzymamy, bezzwlocznie zaczynamy wysylac kolejne pakiety, jesli zostanie przekroczony czas oczekowania to rowniez zaczynamy wysylac pakiety.

```
class SenderThread extends Thread
{
    public void run()
    {

        try
        {
            input = getInputArray();
            while(currentByteIndex < input.size()) {
                int j = currentByteIndex;
                for(int i = 0; i < senderWindow && j < input.size(); i++){
                    Z2Packet p=new Z2Packet(4+1);
                    p.setIntAt(j,0);
                    p.data[4]= input.get(j);
                    DatagramPacket packet =
                            new DatagramPacket(p.data, p.data.length,
                                    localHost, destinationPort);
                    socket.send(packet);
                      System.out.println("*** send package nr: " + j + " : " + ((char)input.ge
                    sleep(sleepTime);
                    j ++;
                }
                receivedConfirmation = false;
                long startTime = System.currentTimeMillis();
                while(!receivedConfirmation && (System.currentTimeMillis()-startTime) < RTT){
                    ;
                }
            }

        }
```

Druga wewnetrzna klasa klasy Z2Sender odpowiada za odbieranie komunikatow od odbiorcy. Nie dzieje sie tutaj nic skomplikowanego. Klasa ma za zadanie odebrac, przetworzyc wiadomosc oraz poinformowac klase wysylajaca pakiety jakie pakiety odbiorca otrzymal lub ewentualnie poinformowac o czasie wysylania wiadomosci jesli dostalismy wiadomosc, ze odbiorca otrzymal juz wszystkie pakiety.

```
class ReceiverThread extends Thread
{

    public void run()
    {
        try
        {
            sendingTime = System.currentTimeMillis();
            while(true)
            {
                byte[] data=new byte[datagramSize];
                DatagramPacket packet=
                        new DatagramPacket(data, datagramSize);
                socket.receive(packet);
                Z2Packet p=new Z2Packet(packet.getData());
                System.out.println("S:"+p.getIntAt(0)+
                        ": "+(char) p.data[4]);

                if(input.size() - 1 >= p.getIntAt(0) && input.get(p.getIntAt(0)) == p.data[4]) {
                    currentByteIndex = p.getIntAt(0) + 1;
                }
                receivedConfirmation = true;
                if(packetNumber == p.getIntAt(0)) {
                    System.out.println("SENDING TIME: " + (System.currentTimeMillis() - sendingTime));
                }
            }
        }
```

Klasa Z2Receiver w przeciwienstwie to klasy Z2Sender operuje na jednym watku. Jesli chodzi o parametry dzialania programu to jest tylko jeden.

```
static final int receiveMessageTime = Z2Sender.senderWindow * Z2Sender.sleepTime;
```

Parametr receiveMessageTime okresla dlugosc czekania na pakiety od nadawcy okreslona od czasu odebrania pierwszego pakietu. Jesli chodzi o dzialanie klasy to dziala ona w taki sposob, ze od czasu odebrania pierwszego pakietu liczymy czas okreslony przez receiveMessageTime I po uplywie tego czasu przesylamy potwierdzenie o odebranych pczkach. Struktura w ktorej zapisujemy odebrane paczki to :

```
private Map<Integer, Byte> receivedPackets = new HashMap<>();
```

Przed wyslaniem potwierdzenia do nadawcy usuwamy z tej struktury wszystkie elementy ktore psuja jego spojnosc ze wzgledu na numery sekwencyjne.

```
class ReceiverThread extends Thread{

    public void run(){
        try
        {
            while(true) {


                long startTime = System.currentTimeMillis();
                handleReceivePocket(startTime);
                startTime = System.currentTimeMillis();
                while((System.currentTimeMillis()-startTime) < receiveMessageTime){
                    handleReceivePocket(startTime);
                }
                // WYSLANIE POTWIERDZENIA
                Z2Packet p=new Z2Packet(4+1);
                receivedPackets = getCoherentMap(receivedPackets);
                if(receivedPackets.size() != 0) {
                    p.setIntAt(Collections.max(receivedPackets.keySet()),0);
                    p.data[4]= receivedPackets.get(Collections.max(receivedPackets.keySet()));
                } else {
                    p.setIntAt(0,0);
                    p.data[4] = 0;
                }

                DatagramPacket packetFoo =
                        new DatagramPacket(p.data, p.data.length,
                                localHost, destinationPort);
                packetFoo.setPort(destinationPort);
                socket.send(packetFoo);
            }
        }
```

Warta omowienia jest funkcja:

```
private void handleReceivePocket(long t) throws IOException {
    try {
        socket.setSoTimeout((int) (receiveMessageTime - (System.currentTimeMillis() - t)));

        byte[] data=new byte[Z2Sender.datagramSize];
        DatagramPacket packet=
                new DatagramPacket(data, Z2Sender.datagramSize);
        socket.receive(packet);
        Z2Packet p=new Z2Packet(packet.getData());
        if(!receivedPackets.keySet().contains(p.getIntAt(0))) {
            receivedPackets.put(p.getIntAt(0), p.data[4]);
            if(hasAllPreviousPackets(p.getIntAt(0))) {
                printNewCoherentPackages(p.getIntAt(0));
            }
        }
    } catch (Exception e) {
    }
```

gdyz to ona odbiera przysylane pakiety I interpretuje czy otrzymane pakietu sa spojne I czy w zwiazku z tym mozna wydrukowac dany pakiet. Po odebraniu pakietu sprawdzamy czy nie mamy juz go zapisanego w celu wykrycia duplikatow, nastepnie sprawdzamy czy ten pakiet jest spojny ze wszystkimi pakietami wystepujacymi przed nim, jesli nie jest to nie drukujemy, a jesli jest, to drukujemy go I wszystkie spojne pakiety wsytepujece przed nim (moglo sie tak zdarzyc, ze byla “dzura” I po dodaniu tego pakietu wiecej pakietow stanie sie spojnych).

Jesli chodzi o szybkosc przesylania pakietow to decydujacy wplyw ma wartosc paramatru RTT. W zwiazku z tym przeprowadzilem testy zwiekszajac wartosci tego parametru liczac czas przeslania wiadomosci. Wyniki testow:

RTT = 20

```
SENDING TIME: 144596
```

RTT = 50

```
SENDING TIME: 148418
```

RTT = 100

```
SENDING TIME: 129991
```

RTT = 200

```
SENDING TIME: 128586
```

RTT = 500

```
SENDING TIME: 127175
```

RTT = 1000

```
SENDING TIME: 148182
```

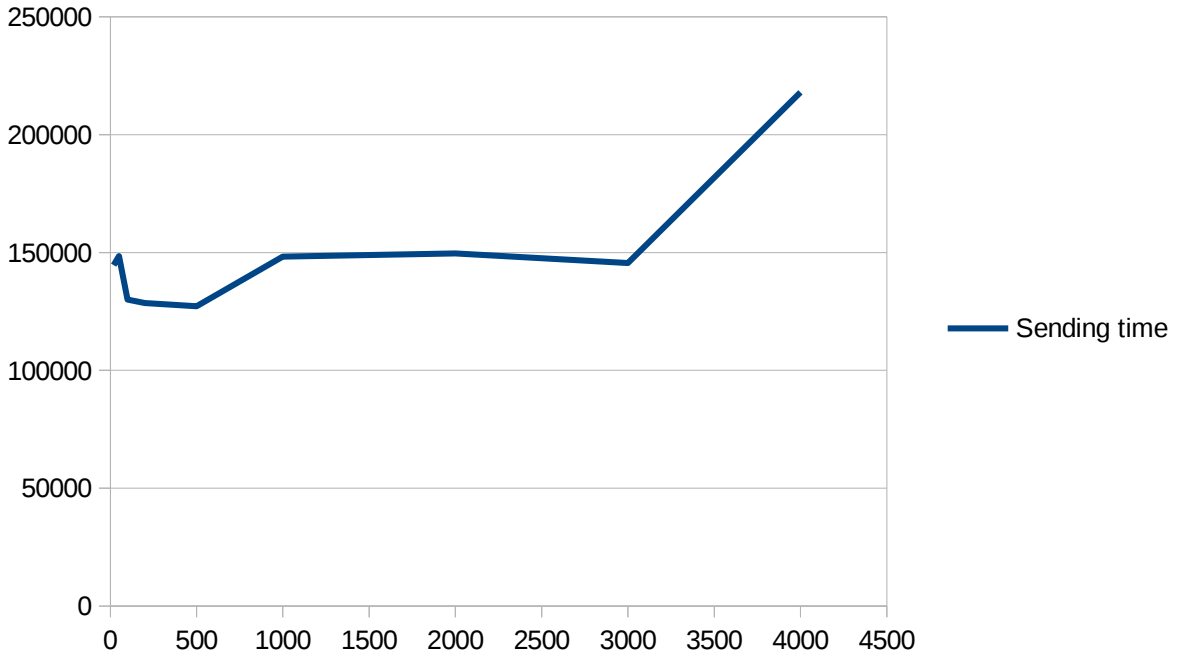
RTT = 2000

```
SENDING TIME: 149593
```

RTT = 4000

```
SENDING TIME: 217941
```

Na wykresie wyniki prezentowalyby sie nastepujaco:



Z otrzymanych wynikow widac, ze znalezienie optymalnego RTT moze sprawiac trudnosci, poniewaz proba znajdoania optymalnego rozwiazania musi bazowac na rozwazaniach heurestycznych. Gdyby testy przeprowadzic bardziej dokladnie okazaloby sie, ze wykres bardziej przypominalby funcje kwadratowa, lecz ze wzlgledu na male wartosci, ciezko bylo zrobic takie testy z duza dokladnoscia. Dla mojego przykladu najbardziej optymalnym wskaznikiem RTT bylaby wartosc 500.