***Цель работы*:** исследование типовых методов и алгоритмов разделения общего ресурса канала между абонентами, определение характеристик рассматриваемых алгоритмов в рамках базовой модели множественного доступа с использованием численных расчетов и имитационного моделирования.

Описание моделируемой системы

Алгоритмы случайного доступа относятся к децентрализованным системам управления доступа к среде. При децентрализованном доступе устройства равноправны и по некоторому алгоритму они организовывают доступ к общему каналу.

Абонент передает сообщение по каналу сообщение готовое к передаче, если по каналу передаются сразу несколько сообщений от разных абонентов, то возникает конфликт, который разрешается по некоторому алгоритму.

Ряд допущений для рассматриваемой модели случайного множественного доступа:

1) Предполагается, что все сообщения у всех абонентов имеют одинаковую длину, время передачи одного сообщения принято за единицу времени. Все время передачи по каналу разбито на окна, длительность окна соответствует времени передачи одного сообщения. Абоненты точно знают моменты разделения и могут начать передачу только в начале окна.

2) В окне возможно 3 события:

- Событие «Конфликт». В окне одновременно передают два абонента или больше. Считается, что из-за наложения сигналов сообщения полностью искажаются и не могут быть приняты правильно.
- Событие «Успех». В окне передает один абонент, в этом случае считается, что абонент успешно передает сообщение.
- Событие «Пусто». В окне никто не передает.

3) Абоненты наблюдают выход канала в конце окна и достоверно определяют, какое из трех событий произошло.

4) В системе имеется M абонентов. В среднем у всех абонентов в одну единицу времени возникает λ сообщений (интенсивность входного потока) (Пуассоновский входной поток с параметром λ). Интенсивность входного потока у всех абонентов в системе одинакова и у каждого абонента она равна λ/M.

Алгоритмом случайного множественного доступа называется правило в соответствии, с которым каждый абонент, имеющий готовое к передаче сообщение, в начале каждого окна решает, передавать сообщение или нет.

Описание исследуемого алгоритма доступа к среде

Рассматривается интервальный адаптивный алгоритм ALOHA. При интервальном варианте алгоритма абонент выбирают номер окна, в котором он будет передавать из интервала $\{1,2,\dots,W_t\}$ при этом параметр W_t изменяется в соответствии с событиями в канале по следующему правилу:

$$W_{t+1} = \begin{cases} \min(2M, 2W_t), \text{"К"} \\ W_t, \text{"У"} \\ \max\left(1, \frac{W_t}{2}\right), \text{"П"} \end{cases}$$

где W_t – размер интервала, в котором абонент выбирает окно для передачи в t-ом окне.

Блок-схема исследуемого алгоритма

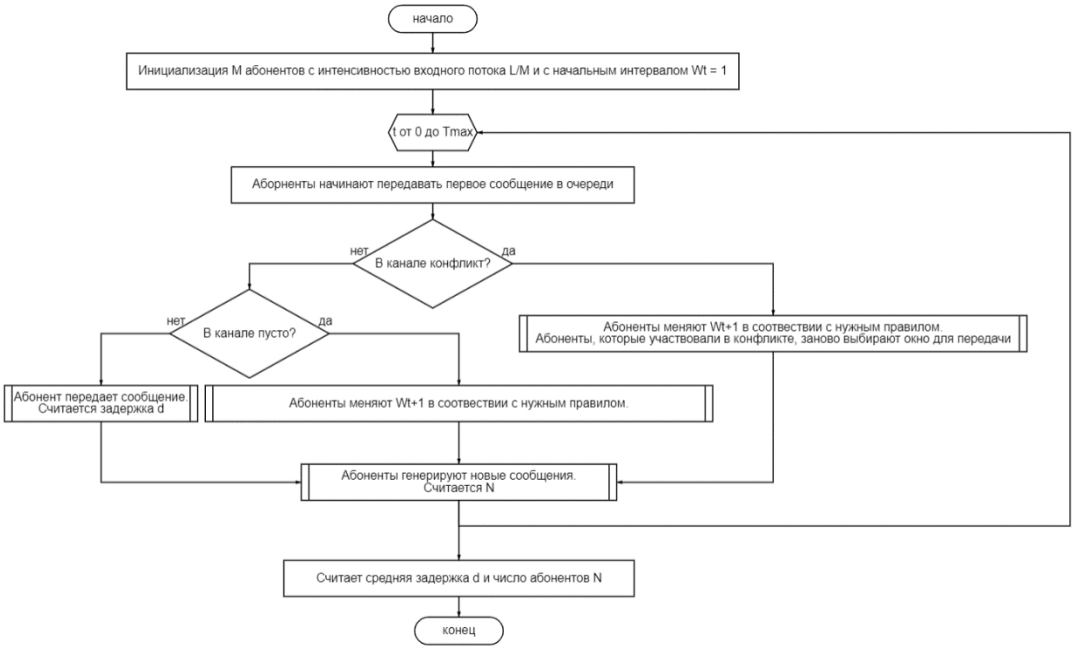

Рисунок 1 - Блок-схема алгоритма

Расчеты значений

Очевидно, что при $M = 1, W_0 = 1 = const$ система будет работать так же, как синхронная система, а значит, теоретические и практические значения $\overline{d}(\lambda_{\text{кр}}), \overline{N}(\lambda_{\text{кр}})$ и $\overline{\lambda}_{\text{вых}}(\lambda_{\text{кр}})$ будут совпадать.

Аналогично можно заметить, что при $W_t = W = const$ система будет работать так же, как интервальный АЛОХА, а, значит значения $\overline{d}(\lambda_{\text{кр}}), \overline{N}(\lambda_{\text{кр}})$ и $\overline{\lambda}_{\text{вых}}(\lambda_{\text{кр}})$ будут совпадать. Можно привести расчет теоретического значения $\lambda_{\text{кр}}$ для данного случая для сравнения со значениями, полученными путем моделирования.

Известно, что для вероятностного алгоритма АЛОХА средняя задержка:

$$d_0(p) = \frac{1}{2} + \frac{1}{p}$$

Также для вероятностного алгоритма АЛОХА критическая интенсивность входного потока определяется как:

$$\lambda_{\text{кр}} = Mp(1-p)^{M-1}$$

Необходимо получить зависимость $p(W)$. Для этого достаточно найти $d_0(W)$ следующим образом:

$$d_0(W) = M[D] = M[D_1] + M[D_2] + M[D_3]$$

где $D_1 -$ время между поступлением сообщения и началом окна, $M[D_1] = \frac{1}{2}$, $D_3 -$ время обслуживания, $M[D_3] = 1$.

$$D_2 = \{0,1, \dots, W-1\}$$

$$\Pr\{D_2 = 0\} = \frac{1}{W}$$

$$\Pr\{D_2 = 1\} = \frac{1}{W}$$

$$\Pr\{D_2 = n\} = \frac{1}{W}$$

Отсюда,

$$M[D_2] = \frac{1}{W}\sum_{0}^{W-1} i = \frac{W-1}{2}$$

Следовательно,

$$d_0(W) = \frac{1}{2} + \frac{W-1}{2} + 1$$

$$d_0(W) = d_0(p) \Rightarrow \frac{1}{2} + \frac{W-1}{2} + 1 = \frac{1}{2} + \frac{1}{p}$$

Отсюда следует, что

$$p = \frac{2}{W+1} \Rightarrow \lambda_{\text{кр}} = M\frac{2}{W+1}\left(1 - \frac{2}{W+1}\right)^{M-1}$$

Графики зависимости

5.1 Сравнение с синхронной системой

Как отмечалось ранее, при $M = 1$ и $W = const = 1$ система работает как синхронная, а значит, результаты можно сравнивать с теоретическими значениями. В этом можно убедиться:

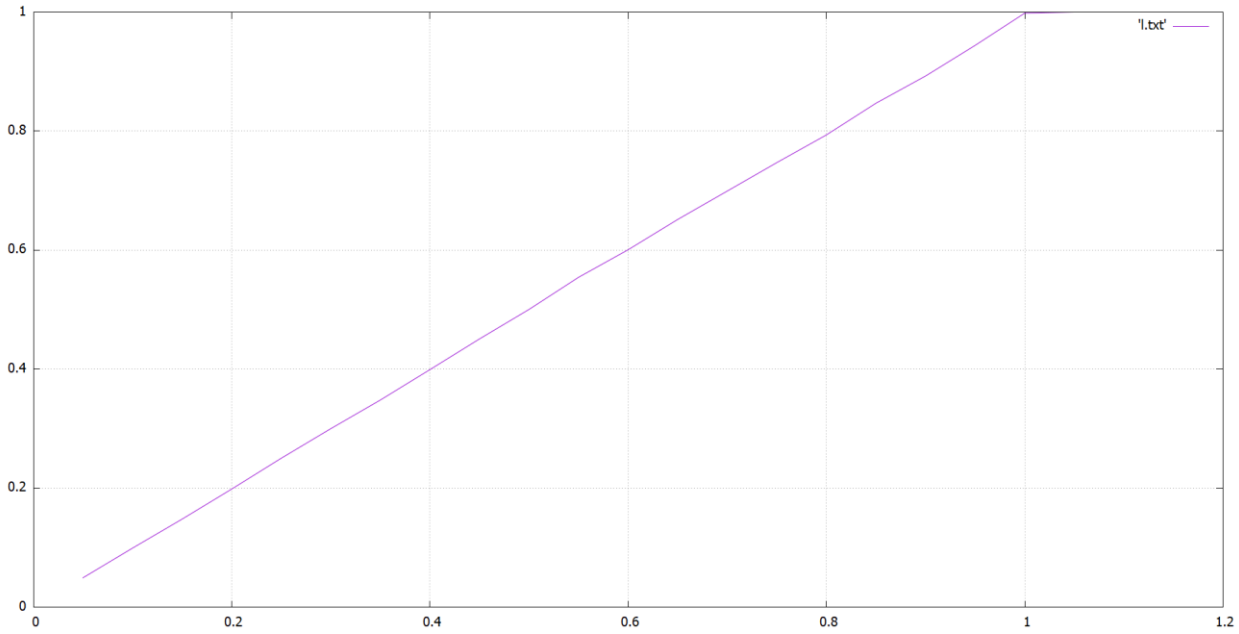

Рисунок 2 – График $\lambda_{\text{вых}}(\lambda)$ для $M = 1, W = 1$

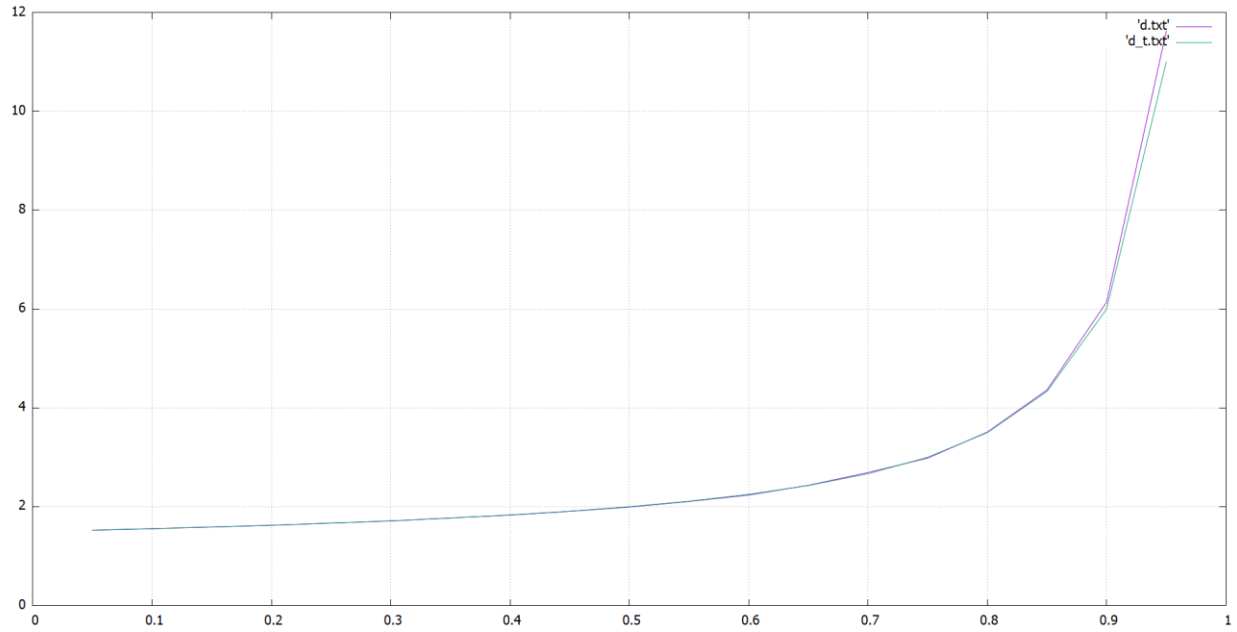

Рисунок 3 – График $d_{\text{теор}}(\lambda)$ и $d_{\text{эксп}}(\lambda)$ для $M = 1, W = 1$

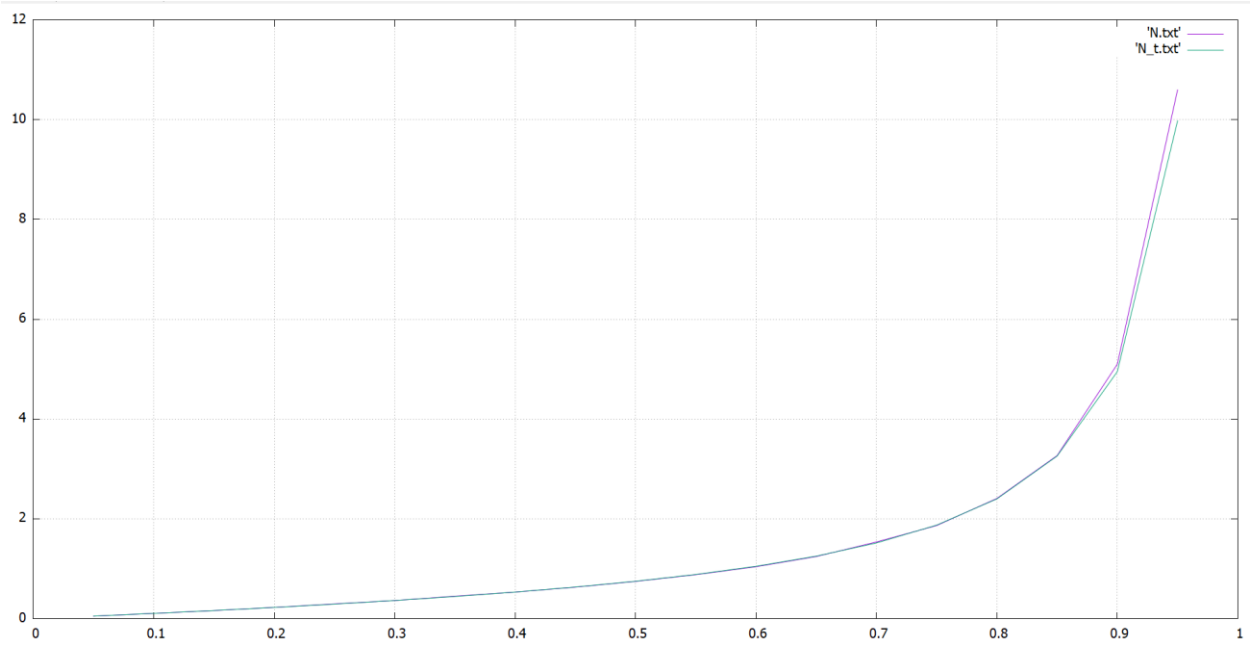

Рисунок 4 – График $N_{\text{теор}}(\lambda)$ и $N_{\text{эксп}}(\lambda)$ для $M = 1, W = 1$

Как видно, графики совпадают.

5.2 Сравнение с интервальным АЛОХА

Как отмечалось ранее, при $W = const$ система работает как интервальный алгоритм АЛОХА, а значит, результаты можно сравнивать с теоретическими значениями. В этом можно убедиться:

Согласно полученной формуле, можно рассчитать $\lambda_{\text{кр}}$ и сравнить с экспериментальной.

Пусть $M = 20$ и $W = 2M$. Тогда $\lambda_{кр} = 0{,}37$ и $d_0 = 21$. Сравним с результатами моделирования:

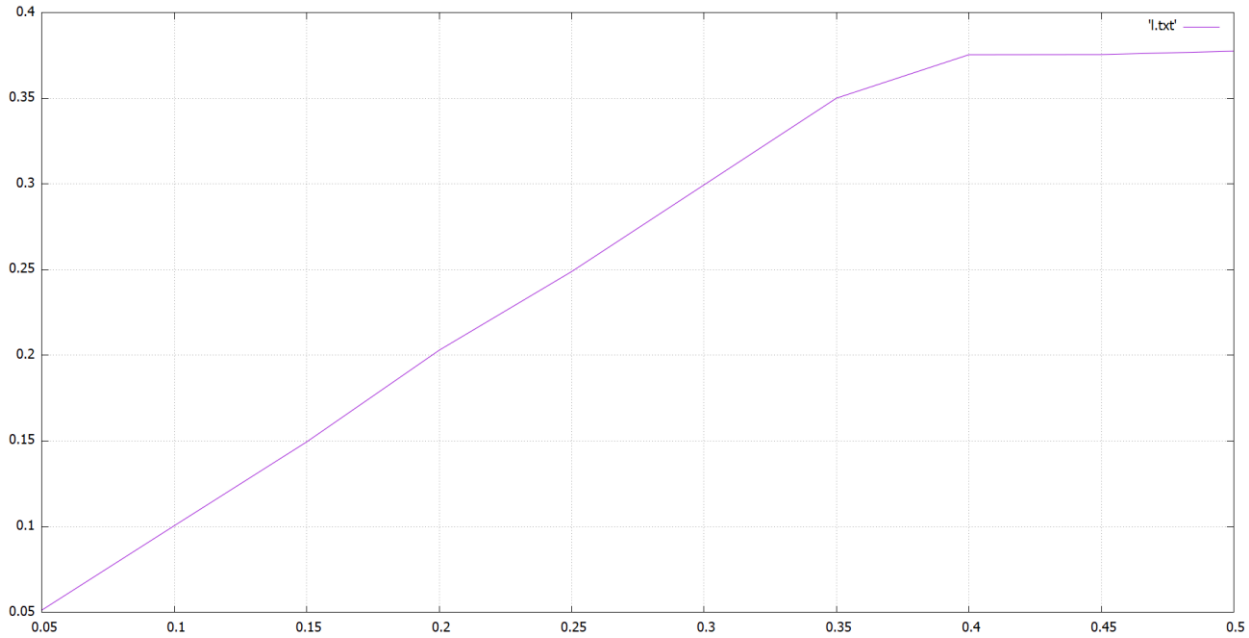

Рисунок 5 – График $\lambda_{вых}(\lambda)$ для $M = 20, W = 40$

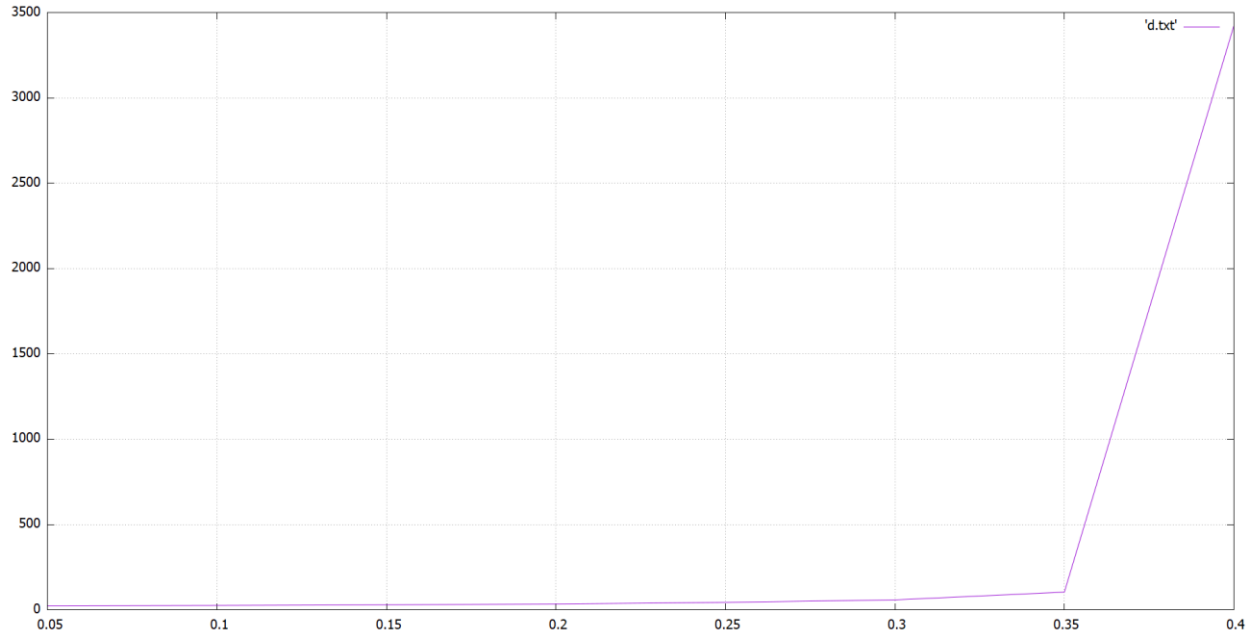

Рисунок 6 – График $d_{эксп}(\lambda)$ для $M = 20, W = 40$

```
lambda = 0.05
lambdaOut = 0.04979
N_e = 1.13115
d_e = 23.21977169812762
```

Рисунок 7 -Значение d_0 для $M = 20, W = 40$

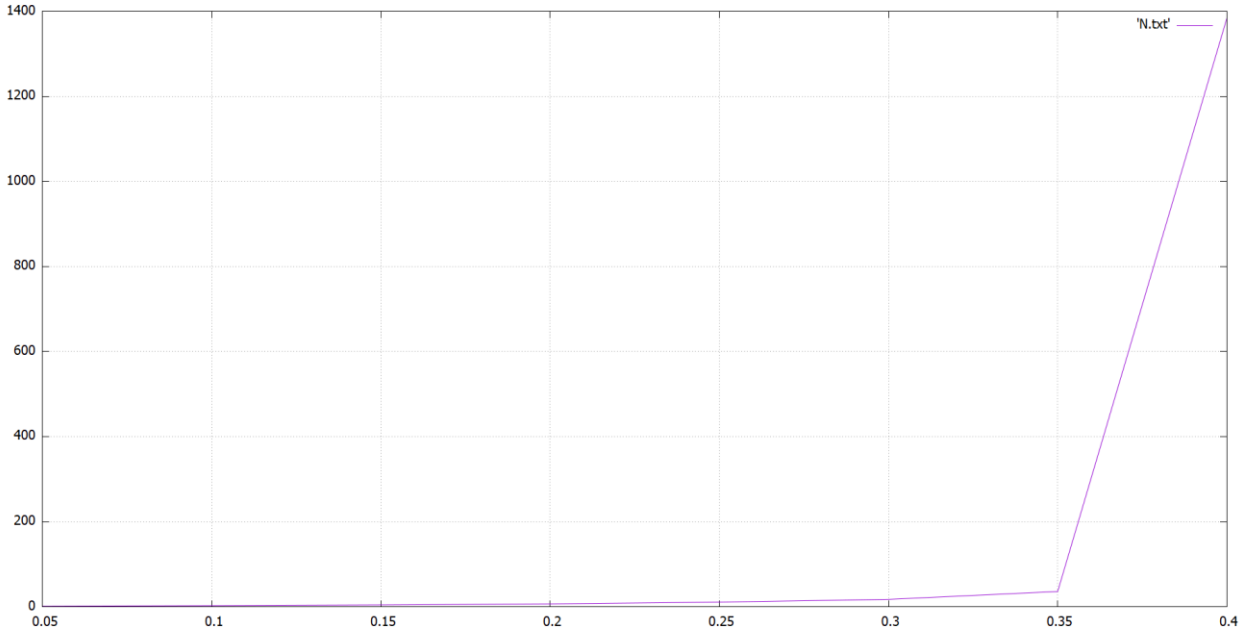

Рисунок 8 – График $N_{\text{эксп}}(\lambda)$ *для* $M = 20, W = 40$

Как видно, при заданных параметрах экспериментальные значения совпадают с теоретическими.

Пусть $M = 10$ и $W = M$. Тогда $\lambda_{\text{кр}} = 0{,}3$ и $d_0 = 6$. Сравним с результатами моделирования:

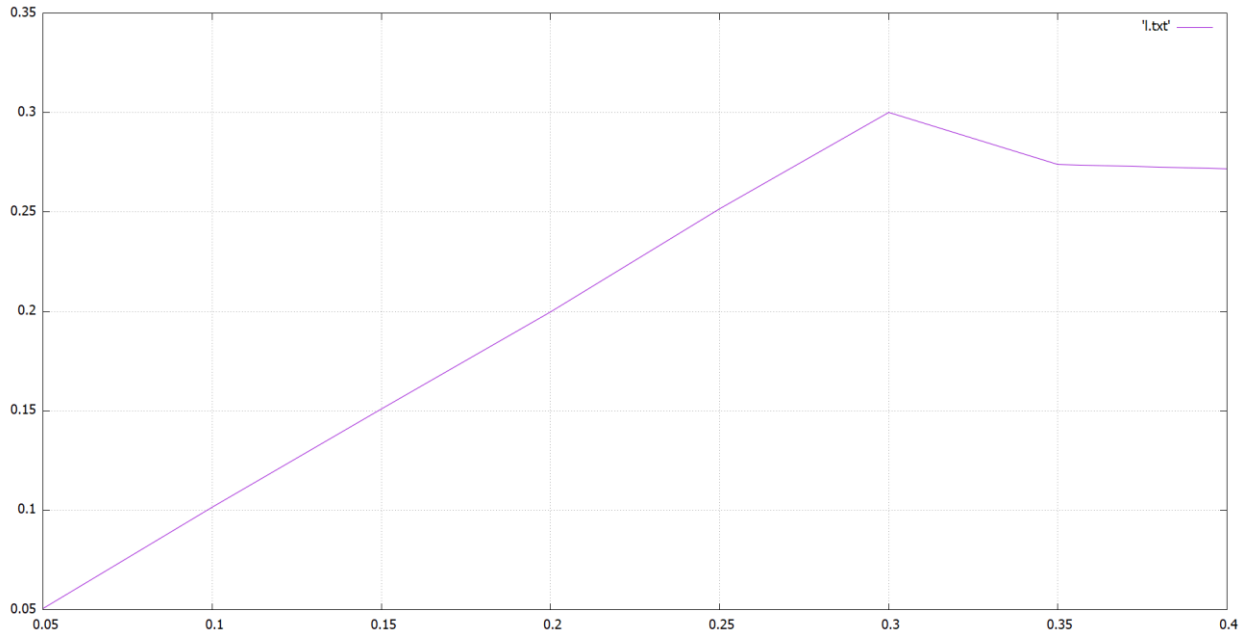

Рисунок 9 – График $\lambda_{\text{вых}}(\lambda)$ *для* $M = 10, W = 10$

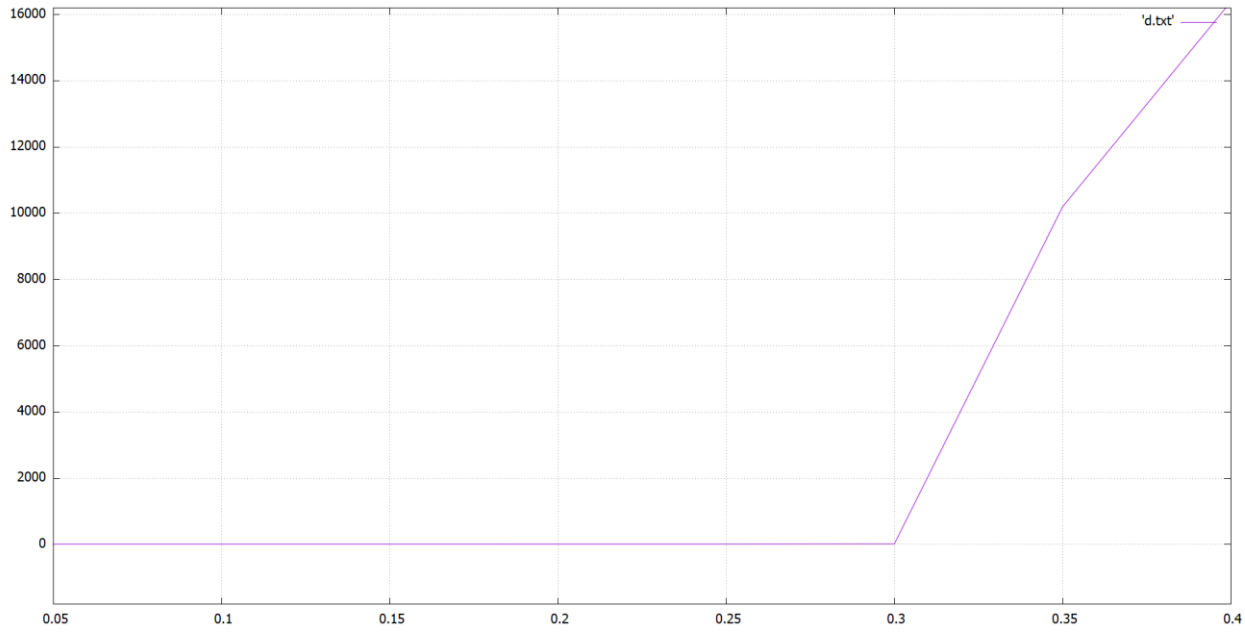

Рисунок 10 – График $d_{\text{эксп}}(\lambda)$ *для* $M = 10, W = 10$

```
lambda = 0.05
lambdaOut = 0.05069
N_e = 0.30084
d_e = 6.4341478195136
```

Рисунок 11 – Значение для $M = 10, W = 10$

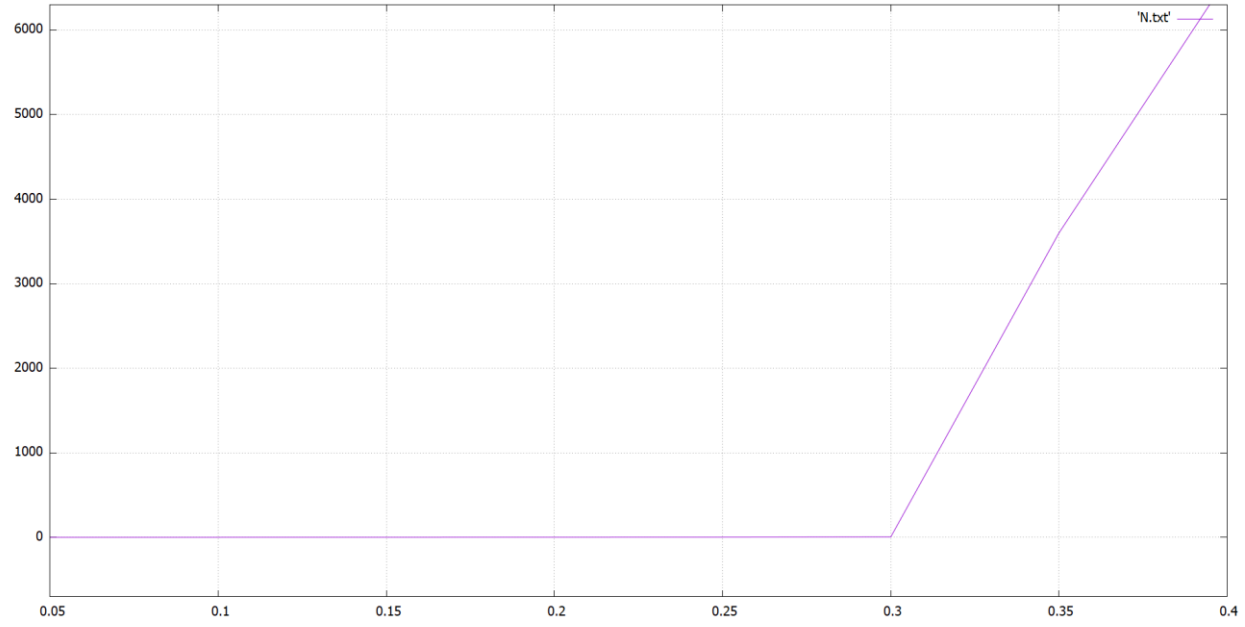

Рисунок 12 – График $N_{\text{эксп}}(\lambda)$ *для* $M = 10, W = 10$

Как видно, при заданных параметрах экспериментальные значения совпадают с теоретическими.

5.3 Исследование интервального адаптивного алгоритма

Пусть $M = 10$ и $W_0 = 20$.

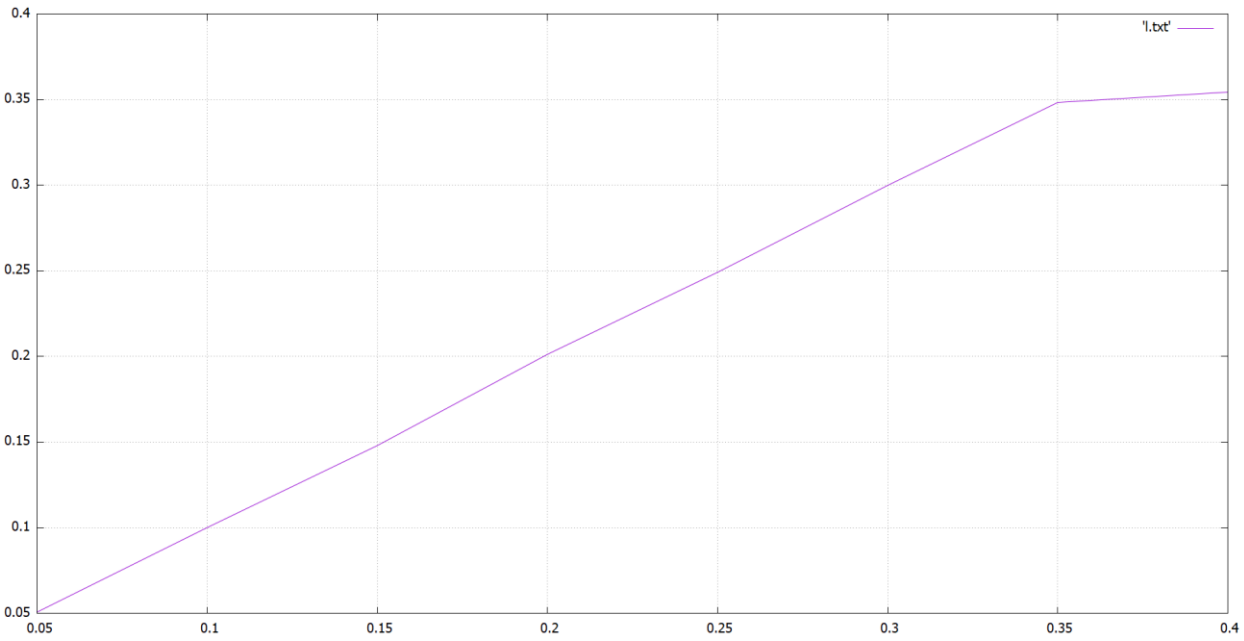

Рисунок 13 – График $\lambda_{\text{вых}}(\lambda)$ для $M = 10, W_0 = 20$

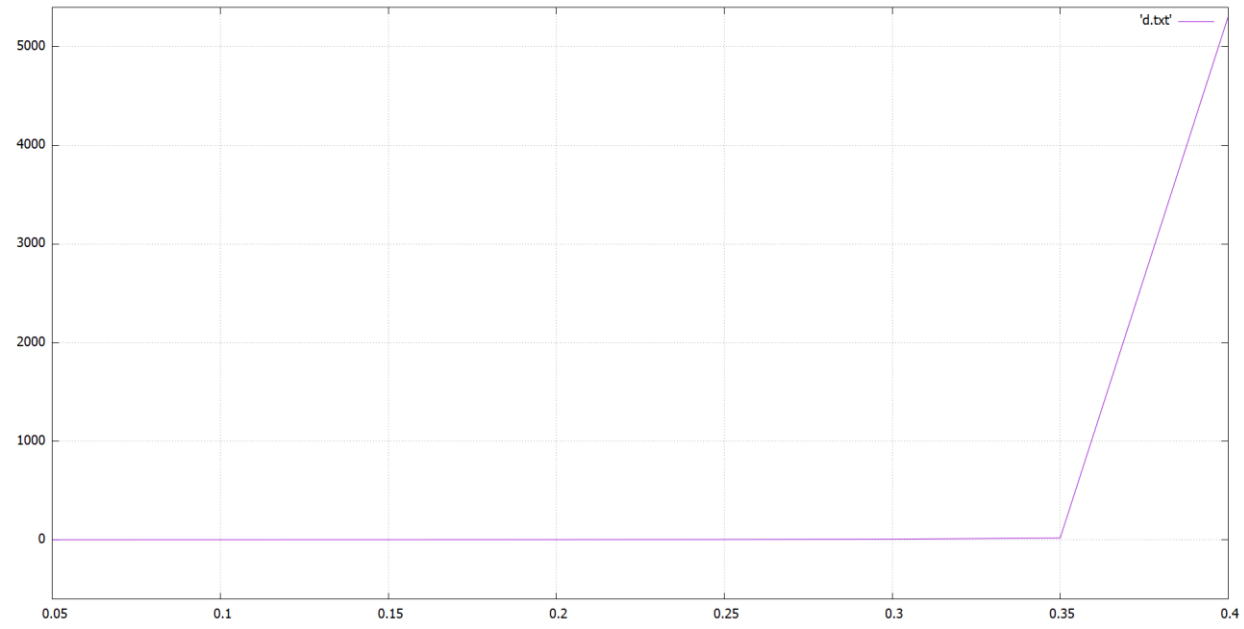

Рисунок 14 – График $d_{\text{эксп}}(\lambda)$ для $M = 10, W_0 = 20$

```
lambda = 0.05
lambdaOut = 0.05078
N_e = 0.0615
d_e = 1.709626020477569
```

Рисунок 15 – Значение d_0 для $M = 10, W_0 = 20$

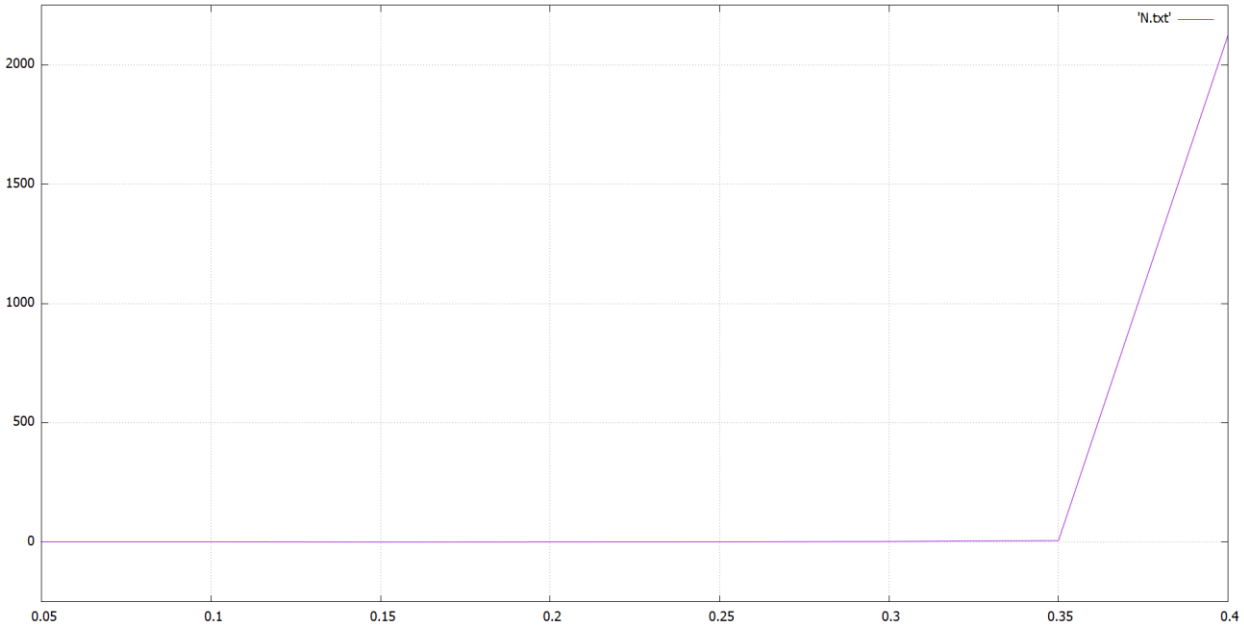

Рисунок 16 – График $N_{эксп}(\lambda)$ для $M = 10, W_0 = 20$

Пусть $M = 20\ W = 20$.

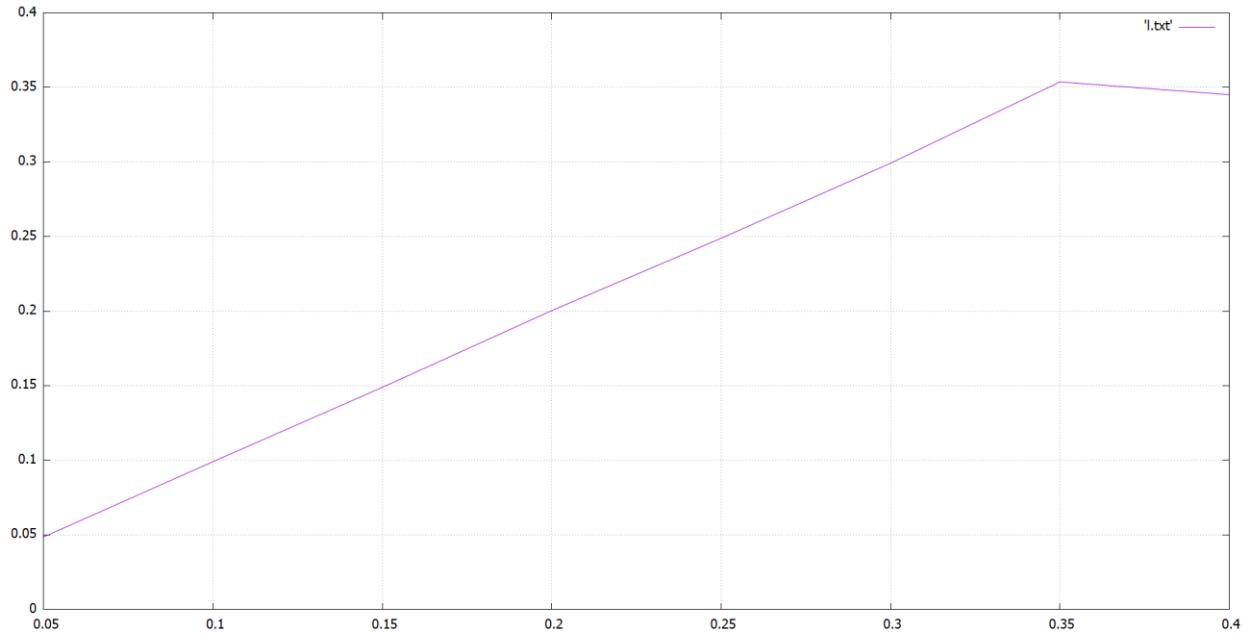

Рисунок 17 – График $\lambda_{вых}(\lambda)$ для $M = 20, W_0 = 20$

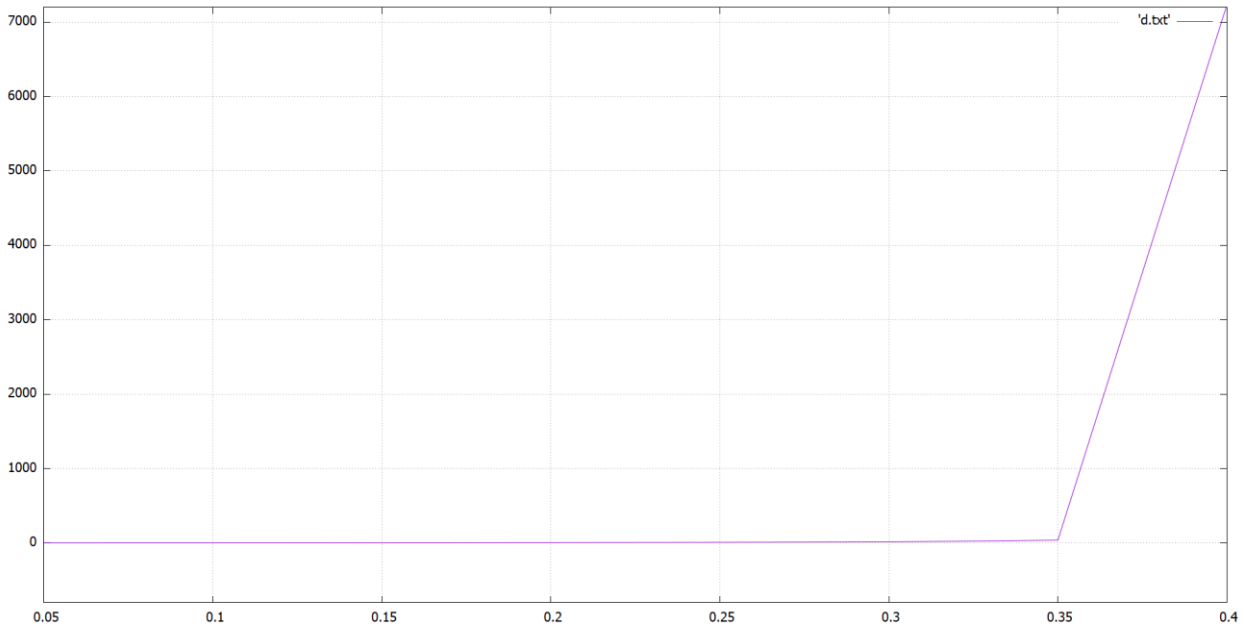

Рисунок 18 – График $d_{\text{эксп}}(\lambda)$ *для* $M = 20, W_0 = 20$

```
lambda = 0.05
lambdaOut = 0.04859
N_e = 0.06207
d_e = 1.7757213895369561
```

Рисунок 19 – Значение d_0 *для* $M = 20, W_0 = 20$

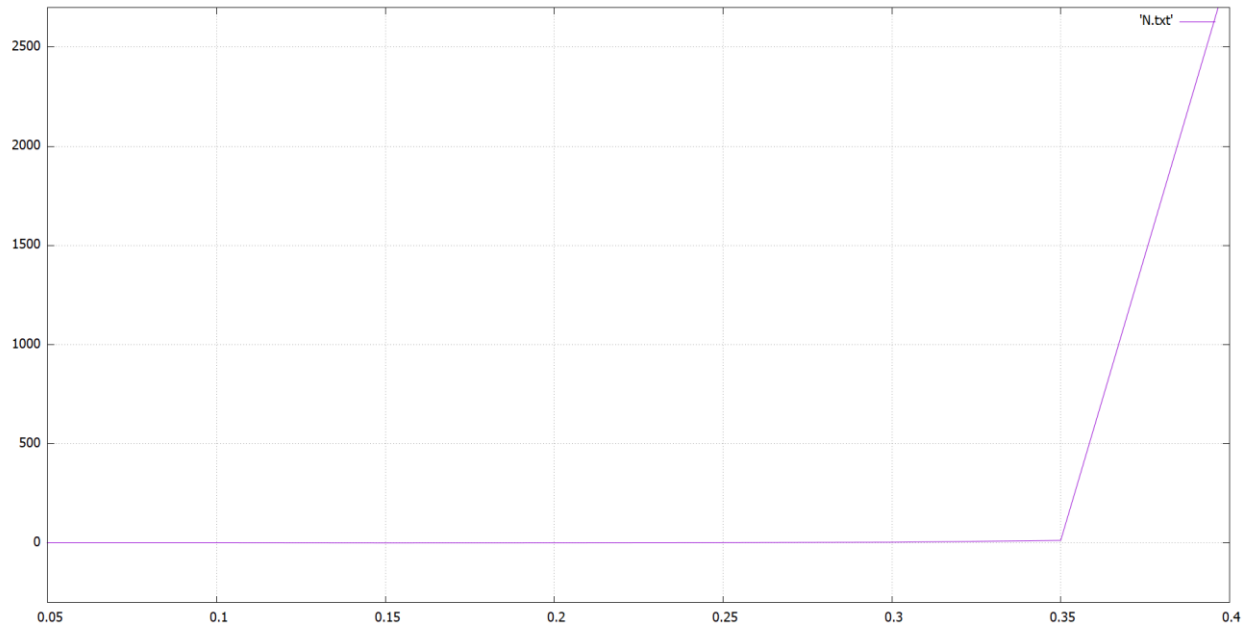

Рисунок 20 – График $N_{\text{эксп}}(\lambda)$ *для* $M = 20, W_0 = 20$

Как видно. При увеличении числа абонентов задержка увеличивается, а и $\lambda_{кр}$ уменьшается.

5.4 Сравнение интервального и интервального адаптивного алгоритма.

Пусть M=20.

Адаптивный интервальный АЛОХА для $W_0 = 1$: $d_0 = 1.86$ и $\lambda_{кр} = 0{,}34$.

```
lambda = 0.05
lambdaOut = 0.04994
N_e = 0.06802
d_e = 1.8631338818695349
```

Рисунок 21 – Значения d_0 для адаптивного интервального

Интервальный АЛОХА для W= $2M$: $d_0 = 22{,}8$ и $\lambda_{кр} = 0{,}37$.

```
lambda = 0.05
lambdaOut = 0.04986
N_e = 1.11688
d_e = 22.895354949342707
```

Рисунок 22 -Значения d_0 для интервального

Таким образом, адаптивный алгоритм обеспечивает меньшую задержку, но $\lambda_{кр}$ меньше. Это можно увидеть и по графикам:

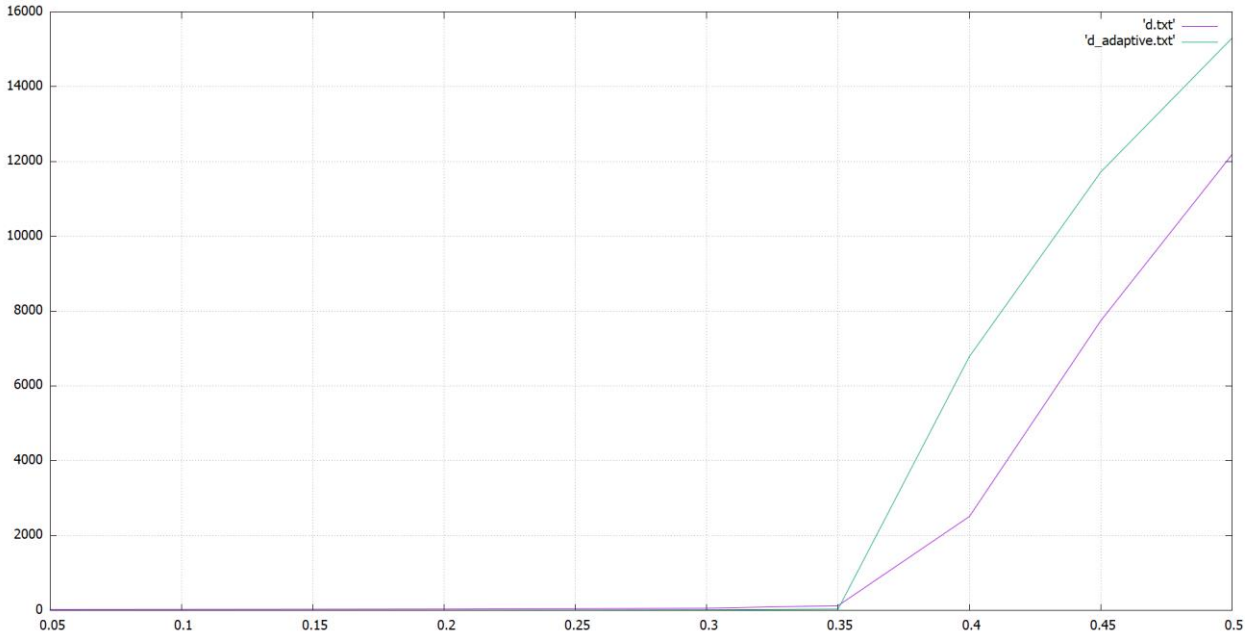

Рисунок 23 - График сравнения

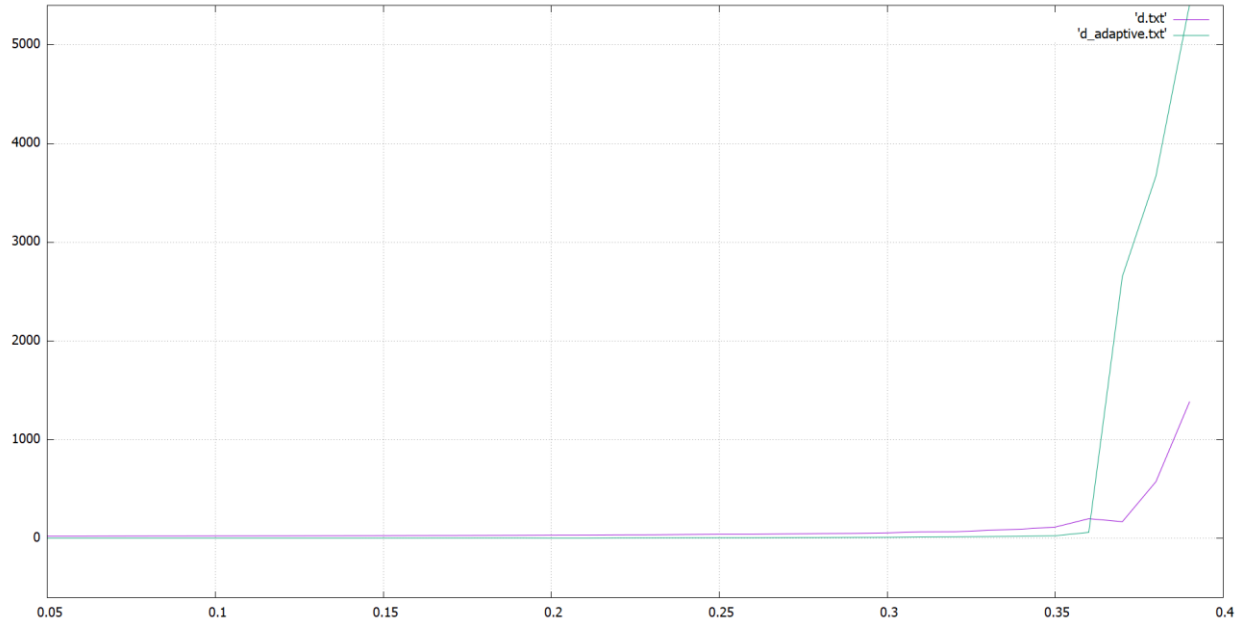

Рисунок 24 - Графики сравнения

Выводы: в ходе выполнения лабораторной работы был исследован и реализован интервальный адаптивный алгоритм АЛОХА, определены характеристики с использованием численных расчетов и имитационного моделирования.

Были построены графики зависимостей $\lambda_{\text{вых}}(\lambda)$, $d(\lambda)$ и $N(\lambda)$. Анализируя графики, можно сказать, что график $\lambda_{\text{вых}}(\lambda)$ представляется собой прямую под углом 45° из начала координат, которая переходит в прямую, параллельную оси OX в точке, советующей $\lambda_{\text{кр}}$. Графики $d(\lambda)$ и $N(\lambda)$ резко возрастают в $\lambda_{\text{кр}}$.

Была рассмотрена работа алгоритма при определенных параметрах в режиме синхронной системы и интервальной АЛОХИ и сделан вывод о том, что графики зависимостей $\lambda_{\text{вых}}(\lambda)$, $d(\lambda)$ и $N(\lambda)$ совпали.

Было произведено сравнение адаптивного и неадаптивного алгоритмов. Был сделан вывод о том, что адаптивный алгоритм обеспечивает меньшую задержку, но $\lambda_{\text{кр}}$ меньше.

Также были построены графики зависимостей для адаптивного интервального АЛОХИ и сделан вывод о том, что с увеличением числа абонентов M критическая интенсивность уменьшается.

Листинг

ALOHA.java

```
package com.suai;

import com.suai.User.Message;
import java.io.FileWriter;
import java.util.ArrayList;


public class ALOHA {

  private double lambda = 1.4; //
  private double T = 100000; // 10000
  private int M = 10;
  private ArrayList<User> users;
  private int Wmax; // МЕНЯТЬ В МОДЕЛИРОВАНИИ


  public ALOHA() {
    users = new ArrayList<>(M);
    for (int i = 0; i < M; i++) {
      users.add(new User(lambda / M));
    }
  }

  public void init() {
    for (int i = 0; i < M; i++) {
      User cur = users.get(i);
      cur.getMessageQueue().clear();
      cur.setLambda(lambda / M);
      cur.setWmax(Wmax);
    }
  }

  private int systemOutput(int window) { // у всех ли надо менять, или
только у участников конфликта
    int readyToSend = 0;
    for (int i = 0; i < users.size(); i++) {
      if (readyToSend >= 2) {
        break;
      }
      User cur = users.get(i);
      if (cur.isReady(window)) {
        readyToSend++;
      }
    }
    if (readyToSend == 0) {
      return 0; // П
    }
```

```
    if (readyToSend == 1) {
      return 1; // У
    }
    return 2; // К
  }

  public void empty() {
    Wmax = (int) (Math.max((1), (double) (Wmax) / 2)); // адаптивный
   // Wmax = (int) (Math.max((Wmax), (double) (Wmax) / 2)); // обычный
интервальный
    for (int i = 0; i < M; i++) {
      users.get(i).empty(Wmax);
    }
  }

  public void conflict(int curWindow) {
    Wmax = Math.min((2 * M), 2 * Wmax); // адаптивный
    //Wmax = Math.min((Wmax), 2 * Wmax); // обычный интервальный
    for (int i = 0; i < M; i++) {
      users.get(i).conflict(Wmax, curWindow);
    }
  }

  private double theoreticalD() {
    double res = theoreticalN() / lambda + 0.5;
    return res;
  }

  private double theoreticalN() {
    double res = (lambda * (2 - lambda)) / (2 * (1 - lambda));
    return res;
  }

  public void channelModeling() {
    for (lambda = 0.05; lambda <= 0.4; lambda += 0.05) {
      Wmax = 20; // МЕНЯТЬ ТУТ
      init();
      double d = 0;
      double N = 0;
      double served = 0;
      for (int t = 0; t < T; t++) {
        int readySub = systemOutput(t);
        if (readySub == 0) {
          for (int i = 0; i < users.size(); i++) {
            User cur = users.get(i);
            empty();
          }
        } else {
          if (readySub == 2) {
            for (int i = 0; i < users.size(); i++) {
```

```
              User cur = users.get(i);
              conflict(t);
            }
          } else {
            for (int i = 0; i < users.size(); i++) {
              User cur = users.get(i);
              if (cur.getMessageQueue().size() != 0) {
                if (cur.isReady(t)) {
                  Message m = cur.getMessageQueue().pollFirst();
                  d += m.getTimeExit() - m.getTimeArrival(); //???
                  served++;
                }
              }
            }
          }
        }
        for (int i = 0; i < M; i++) {
          User cur = users.get(i);
          cur.generateNewMessage(t);
          N += cur.getMessageQueue().size();
        }
      }
      System.out.println("lambda = " + lambda);
      System.out.println("lambdaOut = " + served / T);
      writer("l.txt", lambda, served / T);
      if (M == 1 && Wmax == 1) {
        System.out.println("N_t = " + theoreticalN());
        writer("N_t.txt", lambda,theoreticalN());
      }
      System.out.println("N_e = " + N / T);
      writer("N.txt", lambda, N / T);
      if (M == 1 && Wmax == 1) {
        System.out.println("d_t = " + theoreticalD());
        writer("d_t.txt", lambda, theoreticalD());
      }
      System.out.println("d_e = " + d / served);
      writer("d.txt", lambda, d / served);
    }
  }

  public void cleaner() {
    try {
      String path =
"C:/Users/iyush/IdeaProjects/IntervalAdaptiveALOHA/src/com/suai/";
      FileWriter fileWriterN = new FileWriter(path + "N.txt");
      FileWriter fileWriterD = new FileWriter(path + "d.txt");
      FileWriter fileWriterN_t = new FileWriter(path + "N_t.txt");
      FileWriter fileWriterD_t = new FileWriter(path + "d_t.txt");
      FileWriter fileWriterL = new FileWriter(path + "l.txt");
      fileWriterN.close();
```

```
      fileWriterD.close();
      fileWriterN_t.close();
      fileWriterD_t.close();
      fileWriterL.close();
    } catch (Exception e) {
      System.out.println(e.getMessage());
      e.printStackTrace();
    }

  }

  public void writer(String fileName, double x, double y) {
    try {
      FileWriter file = new FileWriter(

"C:/Users/iyush/IdeaProjects/IntervalAdaptiveALOHA/src/com/suai/" +
fileName, true);
      file.write(x + " " + y + "\n");
      file.flush();
    } catch (Exception e) {
      System.out.println(e.getMessage());
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    ALOHA aloha = new ALOHA();
    aloha.cleaner();
    aloha.channelModeling();
  }
}
```

User.java

```
package com.suai;

import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedList;

public class User {

  public class Message {

    private double timeArrival;
    private double timeExit;

    public double getTimeExit() {
```

```
    return timeExit;
  }

  public double getTimeArrival() {
    return timeArrival;
  }

  public int generateW() { // генерация окна для передачи
    double tmp = Math.random();
    double prob = (double) 1 / Wmax; // равновероятны
    double step = (double) 1 / Wmax;
    int w = 0;
    for (; w < Wmax; w++) {
      if (tmp <= prob)
        break;
      prob += step;
    }
    return w + 1;
  }
}


private HashMap<Integer, Double> prob;
private double lambda;
private LinkedList<Message> messageQueue;
private double Wmax;
private int M;

public LinkedList<Message> getMessageQueue() {
  return messageQueue;
}


public User(double l) {
  prob = new HashMap<>();
  lambda = l;
  messageQueue = new LinkedList<>();
  getProb();
}

public void empty(int Wmax) {
  this.Wmax = Wmax;
}

public void conflict(int Wmax, int curWindows) {
  this.Wmax = Wmax;
  if (messageQueue.size() != 0) {
    if (isReady(curWindows))
    messageQueue.getFirst().timeExit = 0;
  }
```

```
  }

  public void setLambda(double l) {
    lambda = l;
    getProb();
  }

  public void setWmax(double w) {
    Wmax = w;
  }

  public boolean isReady(int curWindow) {
    if (messageQueue.size() != 0) {
      Message first = messageQueue.getFirst();
      if (first.timeExit == 0) {
        first.timeExit = 1 + curWindow + first.generateW() - 1;
      }
    }
    for (int i = 0; i < messageQueue.size(); i++) {
      if (messageQueue.getFirst().timeExit == (curWindow + 1)) { //!!!
        return true;
      }
    }
    return false;
  }


  public long factorial(int number) {  // факториал
    long result = 1;
    for (int factor = 2; factor <= number; factor++) {
      result *= factor;
    }
    return result;
  }

  public double prob(int num) { // подсчет вероятности для i
    double prob = (Math.pow(lambda, num) / factorial(num)) *
Math.exp((-1) * lambda);
    if (prob <= Math.pow(10, -5)) {
      return 0;
    }
    return prob;
  }


  public void getProb() { // генерация вероятностей для числа
абонентов
    for (int count = 0; ; count++) {
      double prob = prob(count);
      if (prob == 0) {
```

```
        break;
      }
      this.prob.put(count, prob);
    }
  }

  public void generateNewMessage(int t) { // генерация числа сообщений
и их запись в очередь
    double tmp = Math.random();
    double sum = 0;
    int newM = 0;
    for (; newM < prob.size(); newM++) {
      sum += prob.get(newM);
      if (tmp <= sum) {
        break;
      }
    }
    for (int k = 0; k < newM; k++) {
      double time = Math.random();
      Message m = new Message();
      m.timeArrival = time + t;
      messageQueue.add(m);
    }
    Collections.sort(messageQueue, new MessageComparator()); //
сортировка по времени выхода
  }

  private class MessageComparator implements Comparator<Message> {

    @Override
    public int compare(Message a, Message b) {
      if (a.getTimeArrival() < b.getTimeArrival()) {
        return -1;
      }
      if (a.getTimeArrival() == b.getTimeArrival()) {
        return 0;
      }
      return 1;
    }
  }
}
```