МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ ИМЕНИ ПАТРИСА ЛУМУМБЫ»

Факультет физико-математических и естественных наук
Кафедра теории вероятностей и кибербезопасности

«Допустить к защите»

Заведующий кафедрой
теории вероятностей
и кибербезопасности
д. т. н., профессор
_____ К. Е. Самуйлов
«___» _____ 20__г.

Выпускная квалификационная работа бакалавра

Направление 02.03.02 «Фундаментальная информатика и информационные технологии»

Тема «Название работы»

Выполнил студент Тагиев Байрам Алтай оглы

Группа НФИбд-02-20
Студенческий билет № 1032200531

Руководитель выпускной
квалификационной работы
профессор кафедры
теории вероятностей
и кибербезопасности
д. ф.-м. н., профессор
Д. С. Кулябов

_____

Автор _____

Москва
2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ ИМЕНИ ПАТРИСА ЛУМУМБЫ»

Аннотация

выпускной квалификационной работы

Тагиева Байрама Алтай оглы

на тему: Название работы

Алгоритм управления очередью, который применяется в маршрутизаторах, играет важную роль в обеспечении качества обслуживания (QoS). В этой работе представляется оценка производительности на основе моделирования и сравнения некоторых популярных методов управления очередями, такие как случайное раннее обнаружение (RED) и DropTail с точки зрения размера очереди, задержки в очереди, изменения задержки в очереди, скорости отбрасывания пакетов и использования полосы пропускания. Моделирование проходит в рамках имитационного и натурного моделирования для понимания эталонной модели и реальных показателей. Результаты моделирования показывают, что алгоритмы семейства RED показывают лучшие результаты в отношении Drop Tail с точки зрения задержки в очереди, изменения задержки в очереди.

Автор ВКР _____ Тагиев Байрам Алтай оглы

Содержание

Перечень условных обозначений и сокращений

PQM — Passive Queue Management — алгоритм пассивного управления очередью

AQM — Active Queue Management — алгоритм активного управления очередью

RED — Random Early Detection — алгоритм случайного раннего обнаружения

ARED — Adaptive RED — самонастраивающийся RED

RARED — Refinde ARED — улучшенный ARED

NS-2 — Network Simulator 2

Введение

Данное исследование посвящено анализу и сравнению алгоритмов управления очередями на основе RED, реализованных с использованием программных средств NS2 и MiniNet. Основная цель исследования - изучить принципы работы и эффективность алгоритмов RED путем моделирования их поведения в различных сетевых условиях. В рамках проекта была предпринята попытка смоделировать поведение сетей с использованием алгоритмов RED и сравнить их производительность с целью определения оптимальных настроек для обеспечения качественной и надежной передачи данных. Результаты моделирования предназначены для определения оптимальных параметров алгоритмов управления очередями, которые могут способствовать общему повышению производительности сетевых систем.

Актуальность темы

Важность этого исследования заключается в изучении механизма активного управления очередями (AQM) в RED, который помогает оптимизировать распределение сетевых ресурсов и обеспечивает соответствие требованиям пользователей к скорости и надежности передачи данных.

Цель работы:

Целью данной выпускной квалификационной работы является исследование и анализ алгоритмов семейства RED (Random Early Detection) для активного управления очередью. Основная задача заключается в изучении принципов работы и эффективности данных алгоритмов в контексте управления потоками данных в сетевых маршутизаторах, а также сравнительный анализ различных алгоритмов в рамках натурной и имитационной модели.

Для достижения поставленных целей будут проведены исследования и анализ различных алгоритмов из семейства RED. В ходе работы будет проанализировано влияние параметров алгоритмов RED на производительность и стабильность сетевых систем, а также будет проведено сравнение эффективности различных вариантов алгоритма RED.

Для достижения целей работы будет использовано программное обеспечение и инструменты моделирования NS2 и Mininet. Основной фокус будет сосредоточен на сравнительном анализе результатов моделирования и выборе оптимальных параметров алгоритма RED для достижения наилучшей производительности.

Основными задачами работы будут:

- Изучение принципов работы алгоритмов семейства RED.
- Сравнительный анализ эффективности различных вариантов алгоритма RED.

- Проведение экспериментов с использованием различных инструментов моделирования для оценки эффективности алгоритмов RED.

В итоге данной работы ожидается получение информации об алгоритмах семейства RED и их применении в сетевых системах, а также оценка эффективности этих алгоритмов с использованием различных инструментов моделирования. Полученные результаты могут быть использованы для оптимизации управления потоками данных в сетевых системах и повышения их производительности и стабильности.

В результате этого исследования мы стремимся получить представление об алгоритмах в рамках RED и их применении в сетевых маршрутизаторах. Мы также оценим эффективность этих алгоритмов с помощью различных инструментов моделирования. Результаты, полученные в результате этого исследования, могут быть использованы для оптимизации управления потоками данных в сетевых системах и повышения их производительности и стабильности.

Краткое содержание работы

Данная работа состоит из введения, трех основных разделов, списка литературы и приложений. В первом разделе рассматриваются инструменты сетевого моделирования и принципы их работы, а также краткий обзор проверяемых алгоритмов. Во втором разделе представлен обзор алгоритмов RED и их реализации в моделях в средствах моделирования NS-2 и Mininet. В третьем разделе подробно представлены результаты моделирования, построены необходимые графики и сделаны выводы относительно эффективности предложенных алгоритма с использованием двух инструментов моделирования, обобщены результаты работы и сделаны основные выводы.

Глава 1. Теоретическое введение

1.1. Методы моделирования

Натурным моделированием называют проведение исследования на настоящем предмете с последующей обработкой результатов опыта на основе теории подобия. Натруное моделирование разделяется на научный эксперимент, комплексные испытания и производственный эксперимент. Научный эксперимент характеризуется обширным применением средств автоматизации, использованием весьма всевозможных средств обработки информации, возможностью вмешательства человека в процесс выполнения эксперимента.

Имитационное моделирование — это способ исследования, при котором исследуемая система сменяется моделью, с достаточной точностью описывающей настоящую систему, с которой проводятся опыты с целью извлечения информации об этой системе. Такую модель можно «проиграть» во времени, как для одного испытания, так и заданного их множества.

1.2. Алгоритмы управления очередями

Алгоритмы управления и обработки очереди разделяются на два основных класса:

Алгоритмы пассивного управления очередью (PQM) — класс алгоритмов, применяемых для обработки очередей, при котором при достижении порогового значения, алгоритм отбрасывает пакеты в соответствии с некоторым правилом конкретной реализации алгоритма. Данные алгоритмы просты в исполнении, однако лишены возможности адаптироваться под разные нагрузки.

Примеры алгоритмов пассивного управления очередью:

- DropTail — отбрасывает пакеты с конца очереди
- DropHead — отбрасывает пакеты с начала очереди
- RandomDrop — отбрасывает случайные пакеты из очереди

Алгоритмы активного управления очередью (AQM) — класс алгоритмов, при

FIXME: добавить сюда про AQM

1.3. Алгоритмы активного управления очередью семейства RED

1.3.1. RED

RED [1] (Random Early Detection, произвольное раннее обнаружение) — алгоритм активного управления очередью (AQM) для управления переполнением очередей маршрутизаторов с возможность предотвращения перегрузок.

Вероятность p_b маркировки на отбрасывание пакетов представляет собой функцию, линейно зависящую от $\hat{q}$, минимального $q_{\min}$ и максимального $q_{\max}$ пороговых значений и параметра $p_{\max}$, определяющего часть отбрасываемых пакетов при достижении средним размером очереди значения $q_{\max}$ и вычисляется следующим образом:

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\min}, \\ \dfrac{\hat{q} - q_{\min}}{q_{\max} - q_{\min}} p_{\max}, & q_{min} < \hat{q} \leqslant q_{max}, \\ 1, & \hat{q} > q_{\max}. \end{cases}$$

1.3.2. Разбор алгоритма RED

Пакет при поступлении в систему попадает в модуль сброса. Решение о сбросе пакета принимается на основе значения вероятности p, получаемого от управляющего модуля. Вероятность p сброса пакетов зависит от экспоненциально взвешенного скользящего среднего размера длины очереди $\hat{q}$, также вычисляемого управляющим модулем, основываясь на текущем значении длины очереди q (см. рис. 1.1).

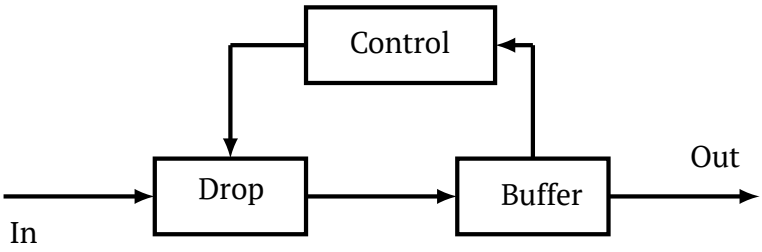

Рис. 1.1.. Модуль RED

График вероятности потери пакета в зависимости от среднего размера очереди приведен на графике 1.2.

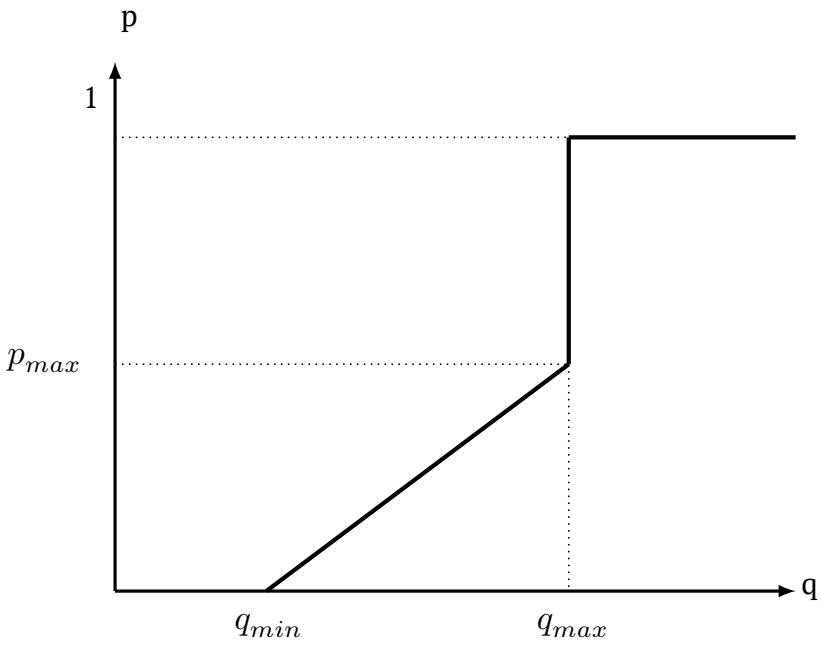

Рис. 1.2.. Вероятность потери пакета в RED

Реализация функции просчета вероятности на отбрасывание пакета в алгоритме RED выглядит следующим образом:

Algorithm 1 RED p calculation

1: **if** $\hat{q} \geq q_{max}$ **then**
2: $p = 1$
3: **else**
4: $p = k * \hat{q} + b$ ▷ график прямой от q_{min} до q_{max}
5: $p* = q_{max}$
6: **end if**

1.3.3. Проблемы RED

Одна из фундаментальных проблем с дизайном RED заключается в том, что он зависит от размера очереди как показателя рабочей нагрузки. В то время как наличие постоянной очереди указывает на состояние перегрузки, длина очереди предоставляет очень мало информации о серьезности состояния перегрузки.

Из-за зависимости алгоритма RED от размера очереди ему присуща проблема точного определения степени перегруженности. Как следствие, RED требует правильных параметров для правильной работы в различных сценариях перегрузки. В то время как RED может

достичь идеального рабочего состояния, это может быть достигнуто только при наличии достаточной буферной емкости и соотминимального порога q_{min}ветствующих настроек параметров.

1.3.4. ARED

В алгоритме Adaptive RED (ARED), разработанном Фенгом [2; 3] и усовершенствованная С. Флойд [4], функция сброса модифицируется посредством изменения по принципу AIMD (принцип AIMD заключается в том, что увеличение некоторой величины производится путём сложения с некоторым параметром, у уменьшение — путём умножения на параметр).

Алгоритм ARED функционирует следующим образом. Для каждого интервала interval (параметр) в секундах, если $\hat{q}$ больше целевого (желаемого) значения $\hat{q_t}$ и $p_{\text{max}} \leqslant 0,5$, то p_{max} увеличивается на некоторую величину α; в противном случае, если $\hat{q}$ меньше целевого значения $\hat{q_t}$ и $p_{\text{max}} \geqslant 0,01$, то p_{max} уменьшается в β раз:

$$p_{max} = \begin{cases} p_{\text{max}} + \alpha, & \hat{q} > \hat{q_t}, \quad p_{\text{max}} \leqslant 0,5, \\ \beta * p_{\text{max}}, & \hat{q} < \hat{q_t}, \quad p_{\text{max}} \geqslant 0,01, \end{cases}$$

где

$$q_{\text{min}} + 0,4\,(q_{\text{max}} - q_{\text{min}}) < \hat{q_t} < q_{\text{min}} + 0,6\,(q_{\text{max}} - q_{\text{min}})\,.$$

Преимущества:

- Автоматическая установка минимального порога q_{min}.
- Автоматическая установка максимального порога q_{max}.
- Адаптивная настройка p_{max}.

1.4. Средства моделирования

1.4.1. Средство имитационного моделирования ns-2

NS-2 (Network Simulator 2) — это симулятор дискретных событий, предназначенный для исследования компьютерных сетей. NS-2 предоставляет существенную поддержку для моделирования протоколов TCP, маршрутизации и многоадресной рассылки по проводным и беспроводным (локальным и спутниковым) сетям.

1.4.2. Средство натурного моделирования Mininet

Mininet — это симулятор сетевых топологий, который позволяет моделировать и анализировать поведение сети в имитируемой среде. Симулятор основан на виртуальных

машинах Linux и технологиях пространства имен, которые используются для создания изолированных сетевых компонентов. Используя Mininet, можно изучать различные сетевые протоколы, алгоритмы маршрутизации и методы управления трафиком. Возможности моделирования Mininet включают создание виртуальных сетевых узлов, настройку топологий (включая связь между узлами и конфигурацию IP-адресов), моделирование различных сетевых условий (таких как задержки, потеря пакетов и пропускная способность) и интеграцию с контроллерами для изучения новых протоколов и алгоритмов.

Глава 2. Моделирование алгоритмов

2.1. Эталонная модель

В рамках анализа алгоритмов была созданна эталонная модель, в рамках которой будут проводиться тестирование. Описание моделируемой сети:

- Сеть состоит из 10 TCP-источников и TCP-приемников, двух маршрутизаторов R1 и R2 между приемниками и источниками.
- Между TCP-источниками и первым маршрутизатором установлена задежрка в 20 мс.
- Между TCP-приемниками и вторым маршрутизатором также установлена задежрка в 20 мс.
- Между маршрутизаторами установлена задежрка в 15 мс. и очередью типа DropTail / RED / ARED (именно здесь и будет происходить сравнение изучаемых алгоритмов).
- Максимальный размер TCP-окна 32; размер передаваемого пакета 1000 байт; время моделирования — 20 секунд.

Данную модель реализовать как в Mininet, так и в NS-2. Сравнение в разных средствах моделирования представлено для понимания того, как модель себя будет вести в идеальной среде (в рамках имитационного моделирования) и в реальной среде (в рамках натурного моделирования).

Реализация данной эталонной модели представлено в прил. А

FIXME: добавить код и комментарии к нему

Заключение

Текст.

В работе было рассмотрено:

1. Принципы работы алгоритмов пассивного и активного управления очередями.
2. Примеры алгоритмов семеймтва RED, преимущества и недостатки.
3. Произведен сравнительный анализ алгоритмов в рамках тестового сценария нагрузки сети, сделан вывод в пользу FIXME: указать алгоритм, который покажет себя лучше.

   FIXME: добавить еще пункты

Итог: FIXME: Добавить сюда результаты

Список литературы

1. *Floyd S.*, *Jacobson V.* Random Early Detection Gateways for Congestion Avoidance // IEEE/ACM Transactions on Networking. — 1993. — Сент. — Т. 1. — С. 397—413. — DOI: 10.1109/90.251892.
2. Techniques for Eliminating Packet Loss in Congested TCP/IP Networks / W.-c. Feng [и др.]. — 1999. — Окт.
3. Self-configuring RED gateway / S.-c. Gateway [и др.] // Proceedings - IEEE INFOCOM. — 2000. — Нояб. — Т. 3.
4. *Floyd S.*, *Gummadi R.*, *Shenker S.* Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. — 2001. — Сент.

Приложение А. Название первого приложения

А.1. Название секции

Текст.

А.2. Название секции

Текст.

Приложение B. Название второго приложения

B.1. Название секции

Текст.

B.2. Название секции

Текст.

В данном приложении представлен исходный код программы для моделирования алгоритмов RED среде Mininet и NS-2.

Приложение C. Заголовочный файл diffur.h

```
/*
    Name: Header file for SDE computing
    Author: Andrew "Atcher" Tchernoivanov
            tchernoivanov@gmail.com
    Copyright: Raccoon Programming Division
*/

# include <stdio.h>
# include <math.h>
# include <gsl/gsl_errno.h>
# include <gsl/gsl_matrix.h>
# include <gsl/gsl_odeiv.h>

# define N 1 // Количество узлов
# define Q_MAX 60 // Максимальное пороговое значение пакетов для
   ↪ алгоритма RED
# define Q_MIN 20 // Минимальное пороговое значение пакетов для
   ↪ алгоритма RED
# define W_MAX 32 // Максимальный размер TCPокна-
# define R 100 // Размер буфера
# define Tp 0.01 // Время прохождения пакета от источника до узла
# define wq 0.0007 // Вес очереди
# define delta 0.01 //
# define C_SMALL 1600 // Количество обслужеваемых за 1 секунду пакетов

double Q_TR_L;
double Q_TR_R;
double Q_TR;
double ALPHA;
double BETA = 0.9;
double BETA_powared = 5; // Compress factor
double P_MAX = 0.1;  // Максимальная вероятность сброса
int K = 3; // Possible values are - 2,3,4

// Описываем нашу индикаторную функцию
double tau (double x)
```

```
{
 if (x > 0.0 )
  return 1.0;
 else
  return 0.0;
}

// Описываем функцию T
double T (double x)
{
 return (Tp+x/(double)C_SMALL);
}

// Описываем функцию C
double C (double x)
{
 if (C_SMALL < x)
  return (double)C_SMALL;
 else
  return x;
}

// Задаем функцию вычисления вероятности сброса
double p_RED (double x) // RED
{
  double p1,p2;

 if ((0.0 <= x) && (x < (double)Q_MIN))
  return 0;
 else if (x > (double)Q_MAX)
  return 1;
 else
  {
   p1 = (double)(x -(double)Q_MIN);
   p2 = (double)((double)Q_MAX - (double)Q_MIN);
   return (double)((p1/p2)*P_MAX);
  }
```

```
}

double p_ARED (double x) // ARED
{
  double p, p1, p2;

// Computing P_MAX
  ALPHA = fmin(0.01, P_MAX/4);

  if ((x > Q_TR) && (P_MAX <= 0.5))
    p = P_MAX+ALPHA;
  else if ((x <= Q_TR) && (P_MAX >= 0.01))
    p = P_MAX*BETA;

  if (p<0.01) P_MAX=0.01;
  if (p>0.5) P_MAX=0.5;
  P_MAX = p;

// Computing p
  if ((0.0 <= x) && (x < (double)Q_MIN))
    return 0;
  else if (x > (double)Q_MAX)
    return 1;
  else
   {
     p1 = (double)(x -(double)Q_MIN);
     p2 = (double)((double)Q_MAX - (double)Q_MIN);
     return (double)((p1/p2)*P_MAX);
   }
}

double p_RARED (double x) // RARED
{
    int a,b;
    double c,d,p,p1,p2;
// Computing P_MAX
    if ((x > Q_TR) && (P_MAX <= 0.5))
```

```
    {
        a = Q_TR - x;
        c = (double) a / (double) Q_TR;
        d = c*P_MAX;
        ALPHA = 0.25 * d;
        p = P_MAX + ALPHA;
    }
    else if ((x <= (double) Q_TR) && (P_MAX >= 0.01))
    {
        a = Q_TR - x;
        b = (int)Q_TR - (int)Q_MIN;
        c = (double)a / (double)b;
        d = 0.17*c;
        BETA = 1 - d;
        p = P_MAX * BETA;
    }
    if (p<0.01) P_MAX = 0.01;
    if (p>0.5) P_MAX = 0.5;
    P_MAX = p;
// Computing P
    if ((0.0 <= x) && (x < (double)Q_MIN))
       return 0;
    else if (x > (double)Q_MAX)
       return 1;
    else
     {
     p1 = (double)(x -(double)Q_MIN);
     p2 = (double)((double)Q_MAX - (double)Q_MIN);
     return (double)((p1/p2)*P_MAX);
     }
}

double p_POWARED (double x) //POWARED
{
    double p, sigma, dev, p1,p2,p3,p4,p11,p21;
    //Computing p_max
    dev = x - Q_TR;
```

```
    if (dev < 0)
    {
        p1 = (double)dev / (double)Q_TR;
        p3 = p1 / (double) BETA;
        p4 = pow(p3,K);
        sigma = fabs(p4);
        p = P_MAX-sigma;
        if ( p < 0) p = 0;
        P_MAX = p;
    }
    else if (dev > 0)
    {
        p11 = (double) R - (double)Q_TR;
        p1 = dev / p11;
        p3 = p1 / (double) BETA;
        p4 = pow(p3,K);
        sigma = fabs(p4);
        p = P_MAX+sigma;
        if ( p > 1 ) p = 1;
        P_MAX = p;
    }
    else if (dev == 0) p = P_MAX;
//Computing p
    if ((0.0 <= x) && (x < (double)Q_MIN))
       return 0;
    else if (x > (double)Q_MAX)
       return 1;
    else
     {
     p1 = (double)(x -(double)Q_MIN);
     p2 = (double)((double)Q_MAX - (double)Q_MIN);
     return (double)((p1/p2)*P_MAX);
     }
}

double W_Reno_RED (double y[])
{
```

```
  return (double)((tau((double)W_MAX-y[0])*(1/T(y[1])))
  +(-((y[0])/2)*(y[0]/T(y[1])))*p_RED(y[2]));
}

double W_Reno_ARED (double y[])
{
  return (double)((tau((double)W_MAX-y[0])*(1/T(y[1])))
  +(-((y[0])/2)*(y[0]/T(y[1])))*p_ARED(y[2]));
}

double W_Reno_RARED (double y[])
{
  return (double)((tau((double)W_MAX-y[0])*(1/T(y[1])))
  +(-((y[0])/2)*(y[0]/T(y[1])))*p_RARED(y[2]));
}

double W_Reno_POWARED (double y[])
{
  return (double)((tau((double)W_MAX-y[0])*(1/T(y[1])))
  +(-((y[0])/2)*(y[0]/T(y[1])))*p_POWARED(y[2]));
}

double W_FReno_RED (double y[])
{
  return (double)((tau((double)W_MAX-y[0])/T(y[1]))
  + (-(((y[0]))*((y[0]))/2)*(1-p_RED(y[2]))
  *p_RED(y[2]) / T(y[1]))+ ((y[0])*(1-(y[0]))*(p_RED(y[2])
  *p_RED(y[2])) / T(y[1])));
}

double W_FReno_ARED (double y[])
{
  return (double)((tau((double)W_MAX-y[0])/T(y[1]))
  + (-(((y[0]))*((y[0]))/2)*(1-p_ARED(y[2]))
 *p_ARED(y[2]) / T(y[1]))+ ((y[0])*(1-(y[0]))
 *(p_ARED(y[2])*p_ARED(y[2])) / T(y[1])));
}
```

```
double W_FReno_RARED (double y[])
{
  return (double)((tau((double)W_MAX-y[0])/T(y[1]))
  + (-(((y[0]))*((y[0]))/2)*(1-p_RARED(y[2]))
  *p_RARED(y[2]) / T(y[1]))+((y[0])*(1-(y[0]))
  *(p_RARED(y[2])*p_RARED(y[2])) / T(y[1])));
}

double W_FReno_POWARED (double y[])
{
  return (double)((tau((double)W_MAX-y[0])/T(y[1]))
  + (-(((y[0]))*((y[0]))/2)*(1-p_POWARED(y[2]))
  *p_POWARED(y[2]) / T(y[1]))+((y[0])*(1-(y[0]))
  *(p_POWARED(y[2])*p_POWARED(y[2])) / T(y[1])));
}

double Q_RED (double y[])
{
  return (double)(-C(y[1])+(tau((double)R-y[1]))
  *(y[0]/T(y[1]))*(1-p_RED(y[2]))*N);
}

double Q_ARED (double y[])
{
  return (double)(-C(y[1])+(tau((double)R-y[1]))
  *(y[0]/T(y[1]))*(1-p_ARED(y[2]))*N);
}

double Q_RARED (double y[])
{
  return (double)(-C(y[1])+(tau((double)R-y[1]))
  *(y[0]/T(y[1]))*(1-p_RARED(y[2]))*N);
}

double Q_POWARED (double y[])
{
```

```
  return (double)(-C(y[1])+(tau((double)R-y[1]))
  *(y[0]/T(y[1]))*(1-p_POWARED(y[2]))*N);
}

double Qe (double y[])
{
  return (double)(((log(1-wq)/delta)*y[2])
  -((log(1-wq)/delta)*y[1]));
}

int func (double t, const double y[], double f[], void *params)
{
  f[0]=W_FReno_ARED(y);
  f[1]=Q_ARED(y);
  f[2]=Qe(y);

 return GSL_SUCCESS;
}
```

Приложение D. Файл diffur.c

```
/*
    Name: Main file for SDE computing
    Author: Andrew "Atcher" Tchernoivanov
            tchernovanov@gmail.com
    Copyright: Raccoon Programming Division
*/

# include <stdio.h>
# include <math.h>
# include <gsl/gsl_errno.h>
# include <gsl/gsl_matrix.h>
# include <gsl/gsl_odeiv.h>
# include "diffur.h"

int main ()
{
  // Задаем границы нашего временного интервала
 double t0 = 0.0, t1 = 200.0;
 // Задаем точку начала отсчета
 double t = t0;
 // и определяем желаемый шаг, с которым у нас будет вычисляться
   ↪ значения
 double h = 1e-3;

// Размерность системы
 int dim_ode = 3;

 // Векторстолбец-, задающий начальные условия
 double y[3] = {1.0, 0.0, 0.0};

 // Определяем метод, который будет использоваться для решения данной
   ↪ системы уравнений
 const gsl_odeiv_step_type *P = gsl_odeiv_step_rk4;

 // Программная: возвращает указатель на начало массива координат
 // для заданного шага и размерности системы
```

```
gsl_odeiv_step *s = gsl_odeiv_step_alloc (P,dim_ode);
// Программная: создание переменной, в которой будет храниться
// накопленная при вычислениях ошибка
gsl_odeiv_control *c = gsl_odeiv_control_y_new (h, t0);
// Программная: возвращает указатель на массив для
// заданной размерности системы
gsl_odeiv_evolve *e = gsl_odeiv_evolve_alloc (dim_ode);

// Определяем нашу общую систему уравнений, передавая
// func - указатель на нашу систему диффуров
// NULL - здесь указывается якобиан, если он есть
// dim_ode - размерность нашей системы уравнений
// NULL - дополнительные параметры, если имеются
gsl_odeiv_system sys = {func, NULL, dim_ode, NULL};

ALPHA = fmin (0.01, P_MAX/4);
Q_TR_L = (Q_MIN + (0.4*(Q_MAX-Q_MIN)));
Q_TR_R = (Q_MIN + (0.6*(Q_MAX-Q_MIN)));
Q_TR = (Q_TR_L + Q_TR_R) / 2;

// Запускаем наш таймер
while (t < t1)
{
  // Считаем значения нашей системы в заданный момент
  // времени при заданных условиях
 int status = gsl_odeiv_evolve_apply (e,c,s,&sys,&t,t1,&h,y);


 if (status != GSL_SUCCESS) // В случае ошибки
   break;                   // прерываем выполнение
 // Выдаем необходимые нам параметры
 printf ("%f %f %f %f\n", t, y[0], y[1], y[2]);
}

// Освобождаем память
gsl_odeiv_evolve_free (e);
gsl_odeiv_control_free (c);
```

```
 gsl_odeiv_step_free (s);

 exit (0);
}
```

Список иллюстраций

Список таблиц