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 к маршрутизатору, играет важную роль в обеспечении качества обслуживания (QoS). В этой работе представляется оценку производительности на основе моделирования и сравнения некоторых популярных методов управления очередями, такие как случайное раннее обнаружение (RED) и DropTail (и другие алгоритмы) с точки зрения размера очереди, задержки в очереди, изменения задержки в очереди, скорости отбрасывания пакетов и использования полосы пропускания. Результаты моделирования показывают, что алгоритмы семейства RED показывают лучшие результаты в отношении Drop Tail с точки зрения задержки в очереди, изменения задержки в очереди и (более низкой) скорости отбрасывания пакетов.

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

Актуальность темы

Текст

Цель работы:

Текст

Краткое содержание работы

Текст

Глава 1. Теоретическое введение

1.1. Методы моделирования

Аналитическое моделирование — процесс формализации реального объекта и нахождение его решения в аналитических функциях. Модель, сформулированная на языке математики, физики, химии или другой науки с помощью системы специализированных символов с точными правилами сочетаемости называется аналитической моделью, чаще всего они представляются в виде формул, неравенств, линейных и нелинейных уравнений, в том числе дифференциальных и интегро-дифференциальных уравнений и их комбинаций. Применяется для анализа характеристик модели, полученной по упрощенным аналитическим зависимостям.

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

FIXME: ((написать про то, зачем эти алгоритмы нужны и где применяются))

Алгоритмы управления и обработки очереди разделяются на два основных класса:

- Passive Queue Management — алгоритм пассивного управления очередью (PQM)
- Passive Queue Management — алгоритм пассивного управления очередью (PQM)

Passive Queue Management — алгоритм 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

Глава 2. Название главы

2.1. Название секции

[1] Random Early Detection — алгоритм случайного раннего обнаружения (RED)

Глава 3. Название главы

3.1. Название секции

Для этого на сервере был запущен виртуальный сервер `xserv`, с IP-адресом `10.130.64.15`:

```
vzctl create 3006  --os template gentoo-x86
vzctl set 3006  --name /xserv  --save
vzctl set 3006  --nameserver 10.130.64.15
vzctl start 3006
vzctl enter 3006
```

Запускаем ssh:

```
/etc/init.d/sshd start
```

Добавим запуск демона ssh по умолчанию:

```
rc-update add sshd default
```

Далее запускаем NX-сервер:

```
nxserver --start
```

Если все в порядке, появляется сообщение:

```
NX> 100 NXSERVER~--- Version 1.4.0-44 OS (GPL)
      NX> 122 Service started
      NX> 999 Bye
```

3.2. Название секции

Текст.

3.3. Название секции

Текст.

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. *Korolkova A.*, *Kulyabov D.*, *Черноиванов А.* К вопросу о классификации алгоритмов RED // Вестник РУДН. Серия «Математика. Информатика. Физика». — 2009. — Янв. — С. 34—46.

Приложение A. Название первого приложения

A.1. Название секции

Текст.

A.2. Название секции

Текст.

Приложение В. Название второго приложения

В.1. Название секции

Текст.

В.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