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 09.03.03 «Прикладная информатика»

Тема «Моделирование систем управления трафиком»

Выполнил студент Саргсян Арам Грачьяевич

Группа НПИбд-01-20
Студенческий билет № 1032201740

Руководитель выпускной
квалификационной работы
доцент кафедры
теории вероятностей
и кибербезопасности
к. ф.-м. н., доцент
А. В. Королькова

_____

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

Саргсяна Арама Грачьяевича

на тему: Моделирование систем управления трафиком

Алгоритмы управления очередью, применяемые в сетевых маршрутизаторах, выполняют ключевую функцию в обеспечении высокого уровня качества обслуживания (Quality of Service, QoS), что в свою очередь способствует эффективному распределению сетевых ресурсов и удовлетворению требований пользователей к задержке, пропускной способности и надежности передачи данных. В рамках выпускной квалификационной работы осуществляется глубокий анализ производительности алгоритма Random Early Detection (RED) и его модификаций через комплексное моделирование. Данный анализ включает в себя сравнение с другими дисциплинами управления очередями с использованием критериев, таких как размер очереди, задержка, вариативность задержки и изменение размера окна протокола TCP. Применение моделирования на базе симулятора NS-2, включающего модификацию исходного кода, и реализация на практике с использованием Mininet, а также инструментов iperf3, tc, netem, позволяют точно оценить производительность и эффективность различных настроек RED. Анализ результатов, полученных с помощью программы Gnuplot, демонстрирует, что алгоритмы семейства RED обеспечивают значительные преимущества по сравнению с традиционным механизмом Drop Tail, особенно в аспектах управления задержкой и ее вариативностью, а также снижения частоты отбрасывания пакетов. Эти выводы подкрепляются количественными данными и графическими иллюстрациями, что делает исследование актуальным для разработчиков сетевого оборудования, стремящихся оптимизировать процессы управления трафиком.

Автор ВКР _____ Саргсян Арам Грачьяевич

Содержание

Введение

Актуальность темы

Текст

Цель работы:

Текст

Краткое содержание работы

Текст

Глава 1. Средства моделирования сетей передачи данных

В данном разделе представим краткий обзор средств моделирования сетей передачи данных.

1.1. NS-2

NS-2 (Network simaulator 2) — это программное средство моделирования сетей, использующееся для исследования и анализа поведения компьютерных сетей. Запуск имитационной модели в данной среде позволяет анализировать различные протоколы и алгоритмы сетевой связи.

NS-2 разработан на языке программирования C++ и TCL, что обеспечивает гибкость и расширяемость средства моделирования. NS-2 содержит библиотеку классов, которые представляют различные элементы сети, такие как узлы, маршрутизаторы, каналы связи и протоколы передачи данных. Для создания модели сети определяются характеристики и параметры каждого элемента сети: пропускная способность канала, задержки, вероятность потери пакетов и другие. После завершения симуляции NS-2 предоставляет мощные инструменты анализа результатов, включая возможность визуализации данных посредством программы NAM (Network animator), статистический анализ и сравнение результатов экспериментов, что позволяет изучать и оценивать производительность различных протоколов и алгоритмов в различных сценариях сети [11; 14].

1.2. Mininet

Mininet [6] — это симулятор сетевых топологий на основе виртуаилизации, который позволяет моделировать и изучать поведение сетей в контролируемой среде, основанный на использовании виртуальных машин и пространств имен Linux для создания изолированных сетевых узлов. Моделирование сетевых топологий с помощью Mininet позволяет исследовать различные сетевые протоколы, маршрутизацию, управление трафиком и т.д. Возможности моделирования с помощью Mininet включают создание виртуальных сетевых узлов, конфигурирование топологий (связь между узлами, настраивать IP-адреса, маршрутизацию), имитировать различные условия сети, такие как задержки, потери пакетов и пропускную способность, интеграция с контроллерами для исследования новых протоколов и алгоритмов.

Некоторые характеристики, которые указали на создание Mininet, включают в себя:

- **Гибкость:** новые топологии и функции могут быть настроены в программном обеспечении с использованием языков программирования и распространенных операционных систем.
- **Применимость:** правильно реализованные прототипы должны быть применимы в реальных сетях на базе оборудования без изменений в исходных кодах.
- **Интерактивность:** управление и запуск симулированной сети должны происходить в режиме реального времени, как если бы это происходило в реальных сетях.
- **Масштабируемость:** среда прототипирования должна масштабироваться до крупных сетей с сотнями или тысячами коммутаторов на одном компьютере.
- **Реализм:** поведение прототипа должно соответствовать реальному поведению с высокой степенью уверенности, чтобы приложения и протоколы могли использоваться без изменений в коде.
- **Возможность совместного использования:** созданные прототипы должны быть легко совместно используемыми с другими сотрудниками, которые могут выполнять и модифицировать эксперименты.

1.3. Cisco Packet Tracer

Packet Tracer — это программное средство, предоставляемое компанией Cisco Systems, позволяющей смоделировать, конфигурировать и отлаживать сетевые сценарии, широко используемое в области сетевых технологий. Данное программное обеспечение предоставляет виртуальную среду, которое позволяет создавать сетевые топологии и настраивать устройства Cisco: маршрутизаторы, коммутаторы и т.д. Графический интерфейс позволяет соединять устройства, устанавливать параметры соединений и задавать настройки протоколов. Cisco Packet Tracer позволяет имитировать передачу данных в сети. Пользователи могут выполнять различные тесты связи, проводить диагностику и мониторинг сетевых устройств, а также создавать и анализировать журналы событий.

1.4. GNS-3

GNS-3 — это программное средство моделирования сетей, позволяющий создавать виртуальные сети, состоящие из реальных или виртуальных устройств, и анализировать их поведение. GNS-3 разработан на языке программирования Python и основан на эмуляторе динамических узлов Dynamips, который позволяет запускать реальные образы операционных систем. В отличие от Packet Tracer, GNS-3 позволяет смоделировать не только

устройства Cisco, но и другие устройства, например, Juniper, Palo, Alto и другие, что позволяет смоделировать различные типы сетей, включая центры обработки данных и облачные инфраструктуры. Одной из главных особенностей GNS-3 является интеграция с виртуальными машинами, что расширяет возможности моделирования. Появляется возможность создавать сетевые сценарии, в которых виртуальные машины выполняют реальные функции, такие как серверы, клиенты, точки доступа Wi-Fi и т.д. Это позволяет проводить натурное моделирование и получить более реалистичные результаты в рамках виртуальной среды.

Глава 2. Обзор дисциплины управлением очередью RED

В данном разделе представим описание работы нескольких алгоритмов семейства RED.

2.1. Линейные модификации

Random Early Detection (RED) — это семейство механизмов предотвращения перегрузки на шлюзе. Он основан на общих принципах, полезен для управления средним размером очереди в сети, где не доверяют взаимодействию между протоколами передачи данных. В отличие от Droptail, который работает таким образом, что когда очередь заполняется, новые пакеты, поступающие в очередь, начинают теряться, алгоритм RED учитывает потоки трафика в сети и стремится предоставить равную пропускную способность для каждого соединения, что позволяет избежать перегрузки сети и улучшить качество обслуживания. В оригинальном RED маршрутизатор вычисляет усредненный по времени средний размер очереди с использованием фильтра нижних частот (экспоненциально взвешенное скользящее среднее) или сглаживания по длине выборки очередей, средний размер очереди сравнивается с двумя пороговыми значениями: минимальным порогом и максимальным. Когда средний размер очереди меньше минимального порога, пакеты не отбрасываются, когда средний размер очереди превышает максимальный порог, отбрасывается все поступающие пакеты. Если размер средней очереди находится между минимальным и максимальным порогом, пакеты отбрасываются с вероятностью p, которая линейно увеличивается до тех пор, пока средняя очередь не достигнет максимального порога. Подробно классический алгоритм описан в [4; 18].

На рисунке **??** представлена упрощенная схема построения современного МЦОВ.

Вероятность p_b маркировки на отбрасывание пакетов представляет собой функцию, линейно зависящую от $\widehat{q}$ (средневзвешенное скользящее среднее), 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 (2.1):

$$p_b = \begin{cases} 0, & 0 < \widehat{q} \leqslant q_{\min}, \\ \frac{\widehat{q} - q_{\min}}{q_{\max} - q_{\min}} p_{\max}, & q_{\min} < \widehat{q} \leqslant q_{\max}, \\ 1, & \widehat{q} > q_{\max}. \end{cases} \tag{2.1}$$

В NS-2 файлы, связанные с RED, прописаны в каталоге `ns-2.35/queue`, там представлены также другие реализации очередей (среди них DropTail, BLUE и т.д.). Следует уделить внимание двум файлам: `red.cc` (исходники), и `red.h` (заголовочный файл). Вероятность отбрасывания пакета прописана в функции

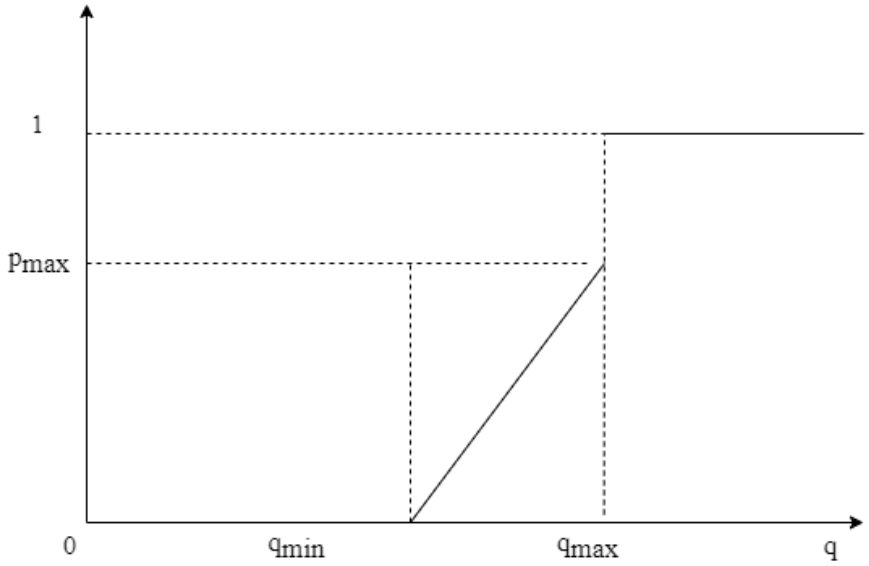

Рис. 2.1.. Вид функции сброса в алгоритме RED

`double REDQueue::calculate_p_ne файла red.cc`

Для реализации в NS-2 необходимо указать в качестве очередей между соединениями RED, и при настройке очереди указать минимальные и максимальные пороговые значения (`thresh_ и maxthresh_`), величина, обратное параметру максимального сброса(`linterm_`), а также указать параметр `gentle_` false.

Для реализации в Mininet используем утилиту tc qdisk … red, имеющий следующие опции:

- min: минимальный порог, по достижении которого возникает вероятность отметки пакета.
- max: максимальный порог очереди
- probability: максимальная вероятность пометки, указанная как число с плавающей точкой, от 0.0 до 1.0.
- limit: жесткий предел реального (не среднего) размера очереди в байтах. По достижении этого размера все лишние пакеты будут отброшены.
- burst: используется для определения того, как реальный размер очереди начинает влиять на средний размер очереди.
- avpkt: указывается в байтах. Используется вместе с burst для определения временной константы для вычисления среднего размера очереди. 1000 - неплохое значение.
- bandwidth: используется для расчета среднего размера очереди после простоя в течение некоторого времени. Должно быть равным значению пропускной способности интерфейса. Не влияет на параметр пропускной скорости интерфейса. Необязательное значение.

Существует несколько причин, по которым существует множество вариаций алгоритмов семейства RED:

1. Разнообразные сетевые сценарии: Разные сетевые сценарии требуют разных настроек и параметров для эффективного управления потоком. Например, алгоритм RED может быть настроен по-разному для использования в локальной сети (LAN) и в глобальной сети (WAN) или в сетях с разной пропускной способностью.

2. Разные типы сетей: RED может быть применен в разных типах сетей, включая проводные и беспроводные сети, и разные типы сетей могут иметь уникальные характеристики и требования, которые влияют на алгоритм.

3. Эволюция сетевых технологий: Сетевые технологии постоянно развиваются, и новые требования и возможности могут потребовать адаптации алгоритма RED. Например, изменения в сетевых протоколах или появление новых типов трафика могут потребовать модификаций алгоритма RED.

4. Эксперименты и исследования: Сетевые исследователи могут создавать различные вариации RED для проведения экспериментов и оценки их производительности в различных условиях.

5. Открытая архитектура: RED - это открытая архитектура, что позволяет исследователям и инженерам создавать свои собственные модификации и адаптации алгоритма в соответствии с конкретными потребностями и задачами.

GRED (Gentle Random Early Detection, мягкое/аккуратное произвольное раннее обнаружение) — алгоритм активного управления очередью, является расширением RED. Стандартный алгоритм увеличивает вероятность отбрасывания с 0.05 до 0.5, когда средняя длина очереди увеличивается от минимального до максимального порогового значения, но при превышении максимального порога вероятность возрастает напрямую с 0.5 до 1. Этот внезапный скачок нормализуется модификацией Gentle RED, который расширяет RED тем, что добавляет дополнительное максимальное пороговое значние, которое равно $2q_{\max}$, тем самым «сглаживая» кривую [12]. Однако, например, задача минимального порога в данной модификации не меняется, и увеличение лишь максимального порога для отбрасывания всех пакетов делает GRED лишь частным случаем классического алгоритма. Данная модификация в NS-2 используется по умолчанию, так как переменная `gentle_` по умолчанию является истинной.

Вероятность сброса определяется по формуле (2.2):

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\min}, \\ \frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}} p_{\max}, & q_{\min} \leqslant \hat{q} < q_{\max}, \\ \frac{\hat{q}-q_{\min}}{q_{\max}}(1-p_{\max}) - p_{\max}, & q_{\max} \leqslant \hat{q} < 2q_{\max}, \\ 1, & \hat{q} \geqslant q_{\max}. \end{cases} \tag{2.2}$$

WRED (Weighted random early detection — взвешенное произвольное раннее обнаружение) — это алгоритм активного управления очередью, является расширением RED [15].

Взвешенный алгоритм произвольного раннего обнаружения предоставляет различные уровни обслуживания пакетов в зависимости от вероятности их отбрасывания и обеспечивает избирательную установку параметров механизма RED на основании типа трафика.

Алгоритм DS-RED(Double slope random early detection) — это ещё одна модификация RED, в котором вводится дополнительное пороговое значение q_{mid} между минимальным q_{min} и максимальным REDФункция сброса описывается двумя линейными сегментами с углами наклона α и β соответственно, регулируемыми задаваемым селектором режимов γ [16]. По умолчанию модификация не реализована в NS-2, для её реализации дополнил фунцию `double REDQueue::calculate_p_ne файла red.cc`, а в программе очереди указал значение переменной `double_slope_ 1`.

Функция вероятности сброса пакетов в алгоритме DSRED показана в формуле (2.3)

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{min}, \\ \alpha\hat{q} - q_{min}, & q_{min} \leqslant \hat{q} < q_{mid}, \\ 1 - \gamma + \beta\hat{q} - q_{mid}, & q_{mid} \leqslant \hat{q} < q_{max}, \\ 1, & \hat{q} \geqslant q_{max}. \end{cases} \tag{2.3}$$

где $\alpha = (\frac{2(1-\gamma)}{\hat{q}-q_{\min}})$, а $\beta = (\frac{2\gamma}{\hat{q}-q_{min}})$

2.2. Нелинейные модификации

Nonlinear RED — это модификация классического алгоритма RED, в котором используется нелинейная функция для определения вероятности отбрасывания пакетов. Nonlinear RED предназначен для более точной адаптации к изменениям трафика и динамике сети. Он способен эффективно реагировать на изменения величины очереди и адаптироваться к различным условиям сети. Это позволяет более гибко управлять задержкой пакетов и предотвращать перегрузки в сети, что делает Nonlinear RED более эффективным по сравнению с классическим алгоритмом RED. [9; 17].

Вероятность p_b маркировки на отбрасывание пакетов вычисляется следующим способом (2.5):

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\min}, \\ 1.5(\frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}})^2 p_{\max}, & q_{\min} < \hat{q} \leqslant q_{\max}, \\ 1, & \hat{q} > q_{\max}. \end{cases} \tag{2.4}$$

По умолчанию NLRED не реализован в NS-2. Для её добавления я использовал патч для данной модификации, созданный Mohit P. Tahiliani для версии 2.34, совместимой также для версии 2.35.

1. Установил к себе на машину патч `NLRED.patch` от
2. Отредактировал файл, заменив везде номер версии на 2.35 и переместил в каталог `ns-allinone`.
3. Дополнил файлы `queue/red.cc`, `queue/red.h`, `tcl/ns-default.tcl` строками из патча, .
4. Переустановил программу.
5. В настройке очереди сети указал значение переменной `nonlinear_ 1`.

HRED(Hyperbola random early detection) —это модификация классического RED с нилейно возрастающей функцией отбрасывания пакетов в сети. HRED менее чувствителен к настройкам параметров, чем другие схемы. При заданном значении $q_{\max}$ HRED ведет себя похожим образом и не сильно зависит от других параметров. HRED может достичь более высокого использования сети. HRED обеспечивает предсказуемую задержку в очереди сети. HRED сохраняет способность контролировать кратковременную перегрузку путем поглощения пакетных потоков, так как он все еще использует алгоритм подсчета среднего размера очереди и поддерживает неполную очередь. Размер очереди может быть задан и зависит от требований. HRED прост в реализации и легко внедряется на маршрутизаторах, так как зменяется только профиль отбрасывания по сравнению с классическим алгоритмом RED [5]. По умолчанию модификация не реализована в NS-2, для её реализации дополнил фунцию `double REDQueue::calculate_p_ne файла red.cc`, а в программе очереди указал значение переменной `hyperbola_ 1`.

Вероятность p_b маркировки на отбрасывание пакетов вычисляется следующим способом (2.5):

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\min}, \\ 1.5(\frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}})^{-1} p_{\max}, & q_{\min} < \hat{q} \leqslant q_{\max}, \\ 1, & \hat{q} > q_{\max}. \end{cases} \tag{2.5}$$

TRED(Three-section random early detection) — это разновидность алгоритма RED, основанный на Nonlinear RED, которая направлена на решение проблем недостаточного использования пропускной способности и больших задержек, возникающих при низкой и высокой нагрузке в RED. Средняя длина очереди TRED между двумя пороговыми значениями разделена на три равные секции, и вероятность отбрасывания пакетов для каждой секции устанавливается по-разному, чтобы адаптироваться к различным трафиковым нагрузкам. С использованием симуляции в среде NS2, TRED эффективно устраняет недостатки RED, увеличивая пропускную способность при низкой нагрузке и снижая задержку при высокой нагрузке. TRED улучшает способность регулировать сетевую перегрузку, повышая использование ресурсов сети и стабильность схемы. В дальнейших исследованиях мы заинтересованы в изучении TRED с явным уведомлением о перегрузке (ECN), поскольку множество исследований показало, что AQM с ECN работает более эффективно, чем без ECN. [3]. По умолчанию модификация не реализована в NS-2, для её реализации дополнил фунцию `double REDQueue::calculate_p_ne файла red.cc`, а в программе очереди указал значение переменной `three_sections_ 1`.

Вероятность p_b маркировки на отбрасывание приведена в (2.6), где $\delta = (q_{max} - q_{min})/3$.

$$p_b = \begin{cases} 0, & 0 \leqslant \hat{q} < q_{\min}, \\ 9(\frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}})^3 p_{\max}, & q_{\min} \leqslant \hat{q} < q_{\min+\delta}, \\ (\frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}}) p_{\max}, & q_{\min} + \delta \leqslant \hat{q} < q_{\min+2\delta}, \\ 9(\frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}})^3 p_{\max} + p_{\max}, & q_{\min} + 2\delta \leqslant \hat{q} < q_{\max}, \\ 1, & \hat{q} \geqslant q_{\max}. \end{cases} \tag{2.6}$$

RED-QL(Random early detection-quadratic linear) —модификация алгоритма RED, также является разновидностью алгоритма с нелинейно возрастающей функцией. RED-QL имеет квадратично-линейную форму и определяется на основе параметров, которые могут быть настроены для определенных требований сети[8] По умолчанию модификация не реализована в NS-2, для её реализации дополнил фунцию `double REDQueue::calculate_p_ne файла re` а в программе очереди указал значение переменной `quadratic_linear_ 1.`.

Вероятность p_b маркировки на отбрасывание приведена в (2.7), где $Target = 2(q_{max} + q_{min})/3 - q_{min}$.

$$p_b = \begin{cases} 0, & 0 \leqslant \hat{q} < q_{\min}, \\ 9(\frac{\hat{q}-q_{min}}{2(q_{\max}-2q_{\min}}))^2 p_{\max}, & q_{\min} \leqslant \hat{q} < Target, \\ p_{max} + 3(1-p_{max})(\frac{\hat{q}-Target}{q_{\max}+q_{\min}}), & Target \leqslant \hat{q} < q_{max}, \\ 1, & \hat{q} \geqslant q_{max}. \end{cases} \tag{2.7}$$

2.3. Адаптивные модификации

В алгоритме Adaptive RED (ARED) функция сброса модифицируется посредством изменения по принципу AIMD, заключающейся в том, что увеличение некоторой величины производится путём сложения с некоторым параметром, у уменьшение — путём умножения на параметр [1]. Для её реализации в NS-2 необходимо указать в настройке очереди `set adaptive_ 1`. Для реализации в Mininet нужно указать в tc дополнительно adaptive

Алгоритм ARED функционирует следующим образом ((2.8)), ((2.9)). Для каждого интервала `interval` (параметр) в секундах, если $\hat{q}$ больше целевой (желаемой) $\hat{q_t}$ и $p_{\max} \leqslant 0,5$, то $p_{\max}$ увеличивается на некоторую величину α; в противном случае, если $\hat{q}$ меньше целевой $\hat{q_t}$ и $p_{\max} \geqslant 0,01$, то $p_{\max}$ уменьшается в β раз, α и β задаются командами `set alpha_` и `set beta_`:

$$p_{\max} = \begin{cases} p_{\max} + \alpha, \ \hat{q} > \hat{q_t}, \ p_{\max} \leqslant 0,5, \\ \beta p_{\max}, \ \hat{q} < \hat{q_t}, \ p_{\max} \geqslant 0,01, \end{cases} \tag{2.8}$$

$$q_{\min} + 0,4(q_{\max} - q_{\min}) < \hat{q_t} < q_{\min} + 0,6\,(q_{\max} - q_{\min})\,. \tag{2.9}$$

Основные особенности:

- автоматическая установка минимального порога $q_{\min}$. Он устанавливается в зависимости от пропускной способности канала C и задержки целевой очереди, q_{max} приравнивается к $3q_{min}$;
- автоматическая настройка w_q. Он устанавливается в зависимости от пропускной способности канала C;
- адаптивная настройка $p_{\max}$. Он адаптирован в соответствии с текущей средней длиной очереди;
- рекомендованными значениями параметров являются $\alpha < 0.25$ и $\beta > 0.83$.

SmRED(Smart random early detection) — модификация RED, в которой вероятность отбрасывания пакетов регулируется в зависимости от нагрузки трафика для достижения

оптимальной сквозной производительности. Кроме того, переход с RED на SmRED в реальной сети требует очень мало работы из-за своей простоты. SmRED эффективно устраняет недостатки RED, увеличивает пропускную способность при низкой нагрузке и уменьшает задержку при высокой нагрузке [2]. По умолчанию модификация не реализована в NS-2, для её реализации дополнил фунцию `double REDQueue::calculate_p_ne файла red.cc`, а в программе очереди указал значение переменной `smart_ 1`.

Вероятность p_b маркировки на отбрасывание приведена в (2.10), где $Target = (q_{max} - q_{min})/2 + q_{min}$.

$$p_b = \begin{cases} 0, & 0 \leqslant \hat{q} < q_{\min}, \\ (\frac{\hat{q}-q_{min}}{q_{\max}-q_{\min}})^2 p_{\max}, & q_{\min} \leqslant \hat{q} < Target, \\ \sqrt{\frac{\hat{q}-q_{min}}{q_{\max}-q_{\min}}} p_{\max}, & Target \leqslant \hat{q} < q_{max}, \\ 1, & \hat{q} \geqslant q_{max}. \end{cases} \tag{2.10}$$

Алгоритм Refined ARED предлагает более активно изменять вероятность сброса $p_{\max}$, чтобы иметь возможность быстрой адаптации к изменяющейся экспоненциально взвешенной скользящей средней длине очереди $\hat{q}$ [7].

Функции изменения параметра $p_{\max}$ представлена ниже((2.11)), ((2.12)):

$$p_{max} = \begin{cases} p_{\max} + \alpha, & \hat{q} > \hat{q}_t, \quad p_{max} \leqslant 0,5, \\ \beta p_{\max}, & \hat{q} \leqslant \hat{q}_t, \quad p_{max} > 0,5, \end{cases} \tag{2.11}$$

$$\begin{cases} q_{\min} + 0,48\,(q_{\max} - q_{\min}) < \hat{q}_t < q_{\min} + 0,52\,(q_{\max} - q_{\min}), \\ \alpha = \left(0,25 \dfrac{\hat{q} - \hat{q}_t}{\hat{q}_t}\right) p_{\max}, \\ \beta = 1 - \left(0,17 \dfrac{\hat{q} - \hat{q}_t}{\hat{q}_t - q_{\min}}\right). \end{cases} \tag{2.12}$$

По умолчанию Refined ARED не реализован в NS-2. Для его добавления я проделал следующие шаги:

1. Установил к себе на машину патч `RARED.patch` от
2. Отредактировал файл, заменив везде номер версии на 2.35 и переместил в каталог `ns-allinone`.
3. Дополнил файлы `queue/red.cc`, `queue/red.h`, `tcl/ns-default.tcl` строками из патча.
4. Переустановил программу.

5. В настройке очереди указал значение `adaptive_ 1` и `refined_adaptive_ 1`.

Powared является модификацией алгоритма ARED [10]. В данной модификации величина p_{max} максимального сброса считается следующим образом((2.13)). Алгоритм POWARED более агрессивно реагирует на изменение средней очереди, чем ARED. Данная модификация не реализована в NS-2, для её моделирования я в файл red.cc добавил функцию `void REDQueue::updateMaxP_powared`, а в настройке очереди сети указал значение переменных `adaptive_ 1` и `powared_ 1`. Параметры модификации задаются с помощью переменных `pwk_, pwb_`.

$$p_{max} = \begin{cases} p_{max} - \delta_1, & q_{\min} \leqslant \hat{q} < q_{mid}, \\ p_{max} + \delta_2, & q_{mid} < \hat{q} \leqslant q_{max}, \\ p_{max}, & \hat{q} = q_{mid}, \end{cases} \tag{2.13}$$

где $q_{mid} = 0.5(q_{min} + q_{max})$, $\delta_1 = |\frac{((\hat{q} - q_{mid})}{(\beta q_{mid}))}|^K$, а $\delta_2 = |\frac{((q_{mid} - \hat{q})}{(\beta(R - q_{mid})))}|^K$.

FARED(Fast Adapting RED) — это алгортитм, который сохраняет целевой диапазон, указанный в алгоритме RARED, но изменяет верхнюю и нижнюю границы для α и β соответственно. Алгоритм FARED обеспечивает надежную производительность в широком диапазоне сред, включая сценарии с умеренной и высокой нагрузкой на трафик [13]. Данная модификация не требует установки каких-либо дополнительных параметров для повышения производительности. Поскольку в алгоритм FARED внесены лишь незначительные изменения по сравнению с ARED и ReARED, он может быть развернут без каких-либо сложностей((2.14), (2.15)). Данная модификация не реализована в NS-2, для её моделирования я в файл red.cc добавил функцию `void REDQueue::updateMaxP_fast_adaptive`, а в настройке очереди сети указал значение переменных `adaptive_ 1` и `fast_adaptive_ 1`.

$$p_{max} = \begin{cases} p_{\max} + \alpha, & \hat{q} > \hat{q}_t, \quad p_{max} \leqslant 0,5, \\ \beta p_{\max}, & \hat{q} \leqslant \hat{q}_t, \quad p_{max} > 0,5, \end{cases} \tag{2.14}$$

$$\begin{cases} q_{\min} + 0,48\,(q_{\max} - q_{\min}) < \hat{q}_t < q_{\min} + 0,52\,(q_{\max} - q_{\min}), \\ \alpha = \left(0,0412 \dfrac{\hat{q} - \hat{q}_t}{\hat{q}_t}\right) p_{\max}, \\ \beta = 1 - \left(0,0385 \dfrac{\hat{q} - \hat{q}_t}{\hat{q}_t - q_{\min}}\right). \end{cases} \tag{2.15}$$

Глава 3. Результаты

3.1. Название секции

Для запуска моделей используем сеть со следующей топологией:

- $N = 20$ TCP-источников, N TCP-приёмников, двух маршрутизаторов $R1$ и $R2$ между источниками и приёмниками (N — не менее 20);
- между TCP-источниками и первым маршрутизатором установлены дуплексные соединения с пропускной способностью 100 Мбит/с и задержкой 20 мс очередью типа DropTail;
- между TCP-приёмниками и вторым маршрутизатором установлены дуплексные соединения с пропускной способностью 100 Мбит/с и задержкой 20 мс очередью типа DropTail;
- между маршрутизаторами установлено симплексное соединение ($R1$–$R2$) с пропускной способностью 20 Мбит/с и задержкой 15 мс очередью типа RED, размером буфера 300 пакетов; в обратную сторону — симплексное соединение ($R2$–$R1$) с пропускной способностью 15 Мбит/с и задержкой 20 мс очередью типа DropTail;
- данные передаются по протоколу FTP поверх TCPReno;
- параметры алгоритма RED: $q_{\min} = 75$, $q_{\max} = 150$, $q_w = 0,002$, $p_{\max} = 0.1$;
- максимальный размер TCP-окна 32; размер передаваемого пакета 500 байт; время моделирования — не менее 20 единиц модельного времени.

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 таких средств моделирования сетей, как NS-2 и Mininet
2. Сделан обзор алгоритма работы дисциплины управления очередью RED и некоторых его модификаций, а также разработаны их реализация в имтационной модели в NS-2.
3. Произведен сравнительный анализ результатов при имитационном и натурном моделировании сети.

Итог: ——.

Список литературы

1. A self-configuring RED gateway / W.-C. Feng [и др.] // IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320). — IEEE, 1999. — DOI: 10.1109/infcom.1999.752150.
2. An AQM based congestion control for eNB RLC in 4G/LTE network / A. K. Paul [и др.] // 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE). — IEEE, 05.2016. — DOI: 10.1109/ccece.2016.7726792.
3. Congestion Control Scheme Performance Analysis Based on Nonlinear RED / C.-W. Feng [и др.] // IEEE Systems Journal. — 2017. — Дек. — Т. 11, № 4. — С. 2247—2254. — DOI: 10.1109/jsyst.2014.2375314.
4. *Floyd S.*, *Jacobson V.* Random early detection gateways for congestion avoidance // IEEE/ACM Transactions on Networking. — 1993. — Т. 1, № 4. — С. 397—413. — DOI: 10.1109/90.251892.
5. *Hu L.*, *Kshemkalyani A.* HRED: a simple and efficient active queue management algorithm // Proceedings. 13th International Conference on Computer Communications and Networks (IEEE Cat. No.04EX969). — IEEE, 2004. — DOI: 10.1109/icccn.2004.1401681.
6. *Keti F.*, *Askar S.* Emulation of Software Defined Networks Using Mininet in Different Simulation Environments // 2015 6th International Conference on Intelligent Systems, Modelling and Simulation. — IEEE, 02.2015. — DOI: 10.1109/isms.2015.46.
7. *Kim T.-H.*, *Lee K.-H.* Refined Adaptive RED in TCP/IP Networks // SICE-ICASE International Joint Conference 2006 Oct. 18-2 1, 2006 in Bexco, Busan, Korea. — IEEE, 2006. — DOI: 10.1109/SICE.2006.314633.
8. *Kumhar D.*, *kumar A.*, *Kewat A.* QRED: an enhancement approach for congestion control in network communications // International Journal of Information Technology. — 2020. — Окт. — Т. 13, № 1. — С. 221—227. — DOI: 10.1007/s41870-020-00538-1.
9. Nonlinear AQM for Multiple RED Routers / L. Lu [и др.] // 2008 Third International Conference on Convergence and Hybrid Information Technology. — IEEE, 11.2008. — DOI: 10.1109/iccit.2008.68.
10. POWARED for Non-Linear Adaptive RED / B. Ng [и др.] // 2005 Asia-Pacific Conference on Communications. — IEEE, 2005. — DOI: 10.1109/apcc.2005.1554179.
11. *Rehmani M. H.*, *Saleem Y.* Network Simulator NS-2 // Encyclopedia of Information Science and Technology, Third Edition. — IGI Global, 07.2014. — С. 6249—6258. — DOI: 10.4018/978-1-4666-5888-2.ch615.

12. Revisiting the Gentle Parameter of the Random Early Detection (RED) for TCP Congestion Control / N. Hamadneh [и др.] // Journal of Communications. — 2019. — С. 229—235. — DOI: 10.12720/jcm.14.3.229-235.

13. *Tahiliani M. P.*, *Shet K. C.*, *Basavaraju T. G.* FARED: Fast Adapting RED Gateways for TCP/IP Networks // IEEE/ACM Transactions on Networking. — 2012. — Т. 1, № 4. — С. 435—443. — DOI: 10.1109/90.251892.

14. *Wei D. X.*, *Cao P.* NS-2 TCP-Linux // Proceeding from the 2006 workshop on ns-2: the IP network simulator - WNS2 '06. — ACM Press, 2006. — DOI: 10.1145/1190455.1190463.

15. Weighted RED (WTRED) Strategy for TCP Congestion Control / N. Hamadneh [и др.] // ICIEIS 2011. Т. II / под ред. A. Abd, M. et al. — Springer, 2011. — С. 421—434. — (Part). — DOI: 10.1007/978-3-642-25453-6_37.

16. *ZHENG B.* DSRED: A New Queue Management Scheme for the Next Generation Internet // IEICE Transactions on Communications. — 2006. — Март. — Т. E89—B, № 3. — С. 764—774. — DOI: 10.1093/ietcom/e89-b.3.764.

17. *Zhou K.*, *Yeung K. L.*, *Li V. O.* Nonlinear RED: A simple yet efficient active queue management scheme // Computer Networks / под ред. C.-T. Lea. — 2006. — 17 мая. — Т. 50, № 18. — С. 3784—3794. — DOI: 10.1016/j.comnet.2006.04.007.

18. *Королькова А. В.*, *Кулябов Д. С.*, *Черноиванов А. И.* К вопросу о классификации алгоритмов RED // Вестник РУДН. Серия «Математика. Информатика. Физика». — 2009. — Янв. — С. 34—46.

Приложение А. Имитационная модель

A.1. main.tcl

```
#новый экземпляр объекта Symulator
set ns [new Simulator]

#трейс файл для nam, файл слишком большой, так что временно закомментируем
set nf [open output/out.nam w]
$ns namtrace-all $nf

#количество источников
set N 20

#создание узлов
source "nodes.tcl"

#метрики TCP
source "TCP.tcl"

#настройка очереди
source "queue.tcl"

#настройка времени моделирования
source "timing.tcl"

#визуализация в nam
source "nam.tcl"

#процедура finish
source "finish.tcl"

#запуск программы
$ns run
```

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 решения стохастических дифференциальных уравнений, написанный на языке C с использоваеним библиотеки GSL.

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