**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**
**«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ»**

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

«Допустить к защите»

Заведующий кафедрой
прикладной информатики
и теории вероятностей
д.т.н., профессор
_____ К.Е. Самуйлов

«____»_____ 20__г.

# Выпускная квалификационная работа бакалавра

Направление 02.03.02 «Фундаментальная информатика и информационные технологии»

ТЕМА «Гибридное моделирование нелинейных систем с управлением в среде OpenModelica»

Выполнил студент **Апреутесей Анна Мария Юрьевна**

(Фамилия, имя, отчество)

Группа НФИбд-01-15

Студ. билет № 1032152610

Руководитель выпускной квалификационной работы

Королькова А.В., к.ф.-м.н., доцент,
доцент кафедры прикладной информатики и теории вероятностей
(Ф.И.О., степень, звание, должность)

_____
(Подпись)

Автор _____
(Подпись)

г. Москва
2019 г.

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Российский университет дружбы народов»**

# АННОТАЦИЯ

**выпускной квалификационной работы**

Апреутесей Анны Марии Юрьевны

(фамилия, имя, отчество)

на тему: Гибридное моделирование нелинейных систем с управлением в среде OpenModelica

Данная работа посвящена применению непрерывно-дискретного подхода к моделированию нелинейных систем с управлением, в качестве которых выступают системы, состоящие из входящего потока, обрабатываемого согласно протоколу Transmission Control Protocol (TCP), а также маршрутизатора, обрабатывающего трафик по алгоритму типа Random Early Detection (RED).

В ходе моделирования в среде OpenModelica был использован гибридный подход, позволяющий учитывать как непрерывные, так и дискретные элементы системы, например, объект управления с непрерывным характером функционирования и дискретное устройство управления. Для реализации вычислительной и имитационной моделей процесса передачи трафика под управлением классического алгоритма RED, а также некоторых его модификаций, применялся язык Modelica, позволяющий проводить гибридное моделирование относительно простым способом.

Автор ВКР _____ _____

(Подпись) (ФИО)

# Оглавление

## Список используемых сокращений

| | |
|---|---|
| ACK | Acknowledgement |
| AQM | Active Queue Management |
| cwnd | Source Congestion Window |
| DDD | Drop Detection Delay |
| DSRED | Double Slope RED |
| ECN | Explicit Congestion Notification |
| ERED | Enhanced RED |
| GRED | Gentle RED |
| RED | Random early detection |
| RTO | Retransmission Time-Out |
| RTT | Round trip time |
| SDRED | State Dependent RED |
| ssthr | Slow Start Threshold |
| TCP | Transmission Control Protocol |

# Введение

Данная работа посвящена применению гибридного подхода к моделированию нелинейной системы с управлением, состоящей из входящего Transmission Control Protocol (TCP) потока и маршрутизатора, обрабатывающего трафик по алгоритму типа Random Early Detection (RED). В ходе работы в среде OpenModelica на языке Modelica были реализованы вычислительная и имитационная модели системы под управлением классического алгоритма RED, а также некоторых его модификаций.

## Актуальность работы

В процессе проектировании телекоммуникационных сетей важно уделить особое внимание математическим исследованиям процесса передачи трафика, так как качественный анализ выбранной политики управления состоянием трафика может предсказать изменение потока в течении времени и улучшить показатели качества обслуживания в сети, что является актуальным вопросом в сфере предоставления услуг связи.

При выборе корректного модельного подхода и средства программной реализации, грамотно отражающей изменения в системе, можно определить способы и добиться улучшения качества обслуживания, что в свою очередь может повысить доходы операторов связи.

Несмотря на очевидные преимущества алгоритма RED, например простота реализации в сетевом оборудовании и эффективное функционирование, в данном алгоритме возникает устойчивый автоколебательный режим функционирования системы при некоторых начальных значениях параметров, что делает поведение системы менее стабильным, и в свою очередь, отрицательно влияет на такие показателя качества обслуживания сети, как пропускная способность, задержка передачи пакетов и т.п.

Гибридное моделирование подобных нелинейных систем дает возможность более точно проанализировать процесс передачи трафика в сети, так как позволяет включить в модель как непрерывные, так и дискретные элементы системы. Среда моделирования OpenModelica дает возможность продемонстрировать простоту создания как математической, так и имитационной моделей гибридной системы с управлением.

Таким образом, актуальным представляется демонстрация и изучение применение гибридного (непрерывно-дискретного) подхода при моделировании систем с управлением, а также исследование алгоритма RED на основе полученной модели, демонстрация возникновения автоколебательного режима при определённых начальных настройках маршрутизаторов.

**Цель работы**

Целью моей работы является демонстрация применения языка Modelica при использовании гибридного или непрерывно-дискретного подхода к моделированию системы передачи трафика с регулируемой алгоритмом типа RED динамической интенсивностью потока.

**Задачи работы**

Основными задачами данной работы являются:

1. Математическое моделирование системы с управлением по алгоритму типа RED и некоторым его модификациям (ERED, DSRED, GRED и SDRED) на языке Modelica;
2. Имитационное моделирование системы с управлением по алгоритму типа RED;
3. Изучение и описание поведения основных параметров системы в зависимости от начальных данных на основе полученных в ходе моделирования результатов;
4. Демонстрация сходств и различий в поведении основных параметров систем, работающих по алгоритму типа RED и его модификациям.

**Методы исследования**

Методом исследования является гибридное моделирование в среде OpenModelica. Язык Modelica используется в качестве языка реализации.

**Апробация работы**

В ходе выполнения работы были получены результаты, представленные на следующих конференциях:

— Всероссийская конференция с международным участием «Информационно-телекоммуникационные технологии и математическое моделирование высокотехнологических систем» (Москва, РУДН, 2018 г., 2019 г.)

— IV Международная научно-практическая конференция «Системы управления, технические системы: устойчивость, стабилизация, пути и методы исследования» (Елец, ЕГУ им. И. А. Бунина, 2018 г.)

— VII Международная конференция «Проблемы математической физики и математического моделирования» (Москва, НИЯУ МИФИ, 2018).

**Публикации**

По теме выпускной квалификационной работы бакалавра были опубликованы работы [1] – [5]. Так же были поданы документы для регистрации программ ЭВМ «Численное моделирование системы с управлением по алгоритму RED» и «Имитационное моделирование гибридной модели протокола TCP» авторов А.М.Ю.Апреутесей и А.В. Корольковой.

**Структура работы**

Работа состоит из введения, трех глав, заключения, списка литературы и приложения.

Во введении отражена проблематика выбранной темы, актуальность данного исследования в современном мире, поставлены цели и задачи данной работы.

В первой части выполнен обзор литературных источников, посвященных теме выпускной работы.

Во второй части представлена аналитическая часть исследования.

Третья часть посвящена математическому и имитационному моделированию в OpenModelica работающих по протоколу TCP систем с управлением по алгоритму RED и его модификациям Enhanced RED, Double Slope RED, Gentle RED и State Dependent RED.

В Заключении описаны результаты и сделаны выводы по проделанной работе, предложены способы использования результатов данного исследования.

В Приложении представлены полные листинги программ, написанных в ходе подготовки Выпускной квалификационной работы.

# 1. Методы и анализ моделирования систем с управлением

## 1.1 Обзор исследований в области моделирования и анализа нелинейных систем с управлением

В приведенных ниже литературных источниках рассматриваются системы с управлением, в которых для построения и дальнейшего анализа процесса передачи трафика применяются различные подходы и методы, такие как, например, методы имитационного и математического моделирования.

В статье [6] описывается принцип функционирования алгоритма RED, который применяется для борьбы с избытком пакетов в очередях маршрутизаторов. В статье говорится, что шлюз обнаруживает перегрузку, вычисляя средний размер очереди. Если длина очереди становится больше некоторого заранее заданного порогового значения, то шлюз маркирует все поступающие в систему пакеты и отбрасывает их с некоторой вероятностью, зависящей от функции среднего размера очереди, тем самым уведомляя источник о перегрузке. В статье представлено математическое описание алгоритма RED и его параметров, рассматривается производительность шлюзов RED для диапазона значений параметров. Шлюзы, использующие алгоритм RED, сохраняют низкий средний размер очереди, благодаря чему в очереди есть возможность для случайных скачков количества пакетов. Шлюзы RED предназначены для сопровождения протокола управления перегрузкой транспортного уровня, таких как TCP. Шлюзы RED не исключают пульсирующее поведение трафика и, уменьшая размер окна, позволяют избежать глобальной синхронизации многих соединений. В статье также описывается, как RED-шлюзы могут использоваться для идентификации тех пользователей, которые используют большую часть полосы пропускания через перегруженный шлюз, также обсуждаются методы эффективной реализации шлюзов RED. Моделирование TCP/IP-сети используются для иллюстрации эффективности шлюзов RED.

В работе [7] авторы проводят моделирование стохастического дифференциального уравнения и анализ поведения размера TCP-окна, была разработана модель для TCP и TCP-подобных протоколов, представлены методики для определения характеристик установившегося состояния производительности потока TCP. Авторы изменяют процесс потери в модели путем обработки потери

событий, поступающие на источник не в виде пакетов, а в виде пуассоновского потока. Это позволило моделировать и анализировать поведение размера TCP окна как управляемый пуассоновский счетчик стохастического дифференциального уравнения. Однако, авторы не берут во внимание некоторые важные детали функционирования протокола TCP, а именно режимы медленного старта, быстрого восстановления, быстрой повторной передачи. Но несмотря на это, отмечается, что полученная методика позволяет проводить анализ жидкостной модели TCP и других механизмах контроля перегрузки.

В работе [8] авторы рассматривают взаимодействие нескольких TCP-потоков с алгоритмом управления RED, разрабатывают динамическую нелинейную модель второго порядка, используя стохастические дифференциальные уравнения. Авторы расширяют представленную в статье [7] модель, устраняют некоторые недостатки, моделируя целостную систему, в которой потери и скорость отправки TCP тесно связаны. Используя инструменты, разработанные в этой статье, авторы представляют критический анализ алгоритма RED, объясняя роль, которую играют параметры конфигурации RED в поведении алгоритма в сети. Для верификации модели проводилось имитационное моделирование с помощью симулятора ns. Авторы отмечают, что представленные методы носят довольно общий характер и могут быть легко расширены для моделирования и анализа других механизмов активного управления очередью (Active Queue Management, AQM).

В статье [9] авторы проанализировали комбинированную модель TCP и AQM с алгоритмом управления типа RED. Авторы использовали линеаризацию для анализа ранее разработанной нелинейной модели TCP, выполнили анализ системы AQM, реализующей RED, и представили рекомендации по выбору параметров системы, которые влияют на ее стабильность. Авторы предложили и разработали два контроллера для активного управления очередью маршрутизаторов, поддерживающих TCP потоки с использованием ранее разработанной линеаризованной модели TCP и AQM. Системы, работающие под управлением P-контроллера и PI-контроллера, имели более быстрый отклик и лучшие теоретические свойства, по сравнению с известным RED контроллером. Авторы отмечают, что модель с пропорциональным контроллером показала хороший отклик на изменения потока, но пострадала от стационарных ошибок в регулировании

очереди, что негативно отразится на системах, где размер буфера ограничен. Руководствуясь этим ограничением, авторы разработали классический PI-регулятор, который демонстрировал нулевую погрешность в установившемся режиме и более быстрый отклик на изменения потока. PI-контроллер показал лучшую производительность во всех смоделированных авторами случаях, также было продемонстрированно влияние PI-контроллера на управление очередями и задержкой. Для нелинейного моделирования систем и отслеживания производительности контроллеров авторами использовалось средство моделирования ns.

В статье [10] о жидкостных моделях и решениях для крупномасштабных IP-сетей авторы описывают масштабируемую модель сети AQM маршрутизаторов, которая необходима для обслуживания большого количества TCP потоков. Авторами представлены эффективные методы решения, позволяющие получить промежуточные значения таких параметров сети, как значение средней длины очереди, задержки, вероятности потери пакетов и среднюю пропускную способность сети. В статье моделируются поведение ряда вариантов TCP, таких как SACK, Reno и Newreno, с различными механизмами AQM, такими как RED, AVQ [11] и управление PI-контроллером [9]. Авторы опираются на выведенную в статье [8] модель, которая отражает поведение TCP сетей с помощью системы обыкновенных дифференциальных уравнений, описывающих ожидаемое или среднее поведение системы. Авторы также указывают на ряд недостатков в публикации [7]: в исходной модели определена матрица трафика, описывающая набор маршрутизаторов, через которые проходит определенный набор потоков, однако порядок, в котором потоки проходили через маршрутизаторы, отсутствовал в матрице трафика, и эта информация, по мнению авторов, потенциально имеет критическое значение. В работе [10] авторы включают топологическую информацию в систему дифференциальных уравнений. В качестве средства моделирования используется ns симуляция, результаты которой показывают, что представленные авторами модели достаточно точны и в то же время требуют значительно меньше времени, чем моделирование на уровне пакетов, особенно в сетях, где нагрузки и пропускная способность высоки. Представленное решение обеспечивает довольно простое распараллеливание, а описанные модели хорошо

масштабируются при увеличении размера сети, демонстрируя линейный рост вычислительной сложности.

В работе [12] рассматриваются и расширяются некоторые аспекты теории систем массового обслуживания с обратной связью, применяется ряд моделей для анализа производительности протокола управления передачей, протокола управления потоком. Автор рассматривает жидкостные модели массового обслуживания для случайного пуассоновского процесса с непрерывным временем. Проводится моделирование в сетевом симуляторе ns-2 реакции источников на потери пакетов, возникающие при переполнении буфера, как при т.н. «синхронных потерях» (все источники снижают свою скорость после перегрузки буфера маршрутизатора), так и при «пропорциональных потерях» (только один из источников снижает свою скорость).

В работе [13] автор изучает проблему регулирования механизма формирования окна в системах, работающих согласно TCP-подобным протоколам с организацией обработки очередей по алгоритму RED. В ходе работы автор поднимает вопрос поиска решения, способного обеспечить хорошее качество обслуживания даже при условиях, не являющимися стандартными сетевыми настройками RED. Решая данную проблему, автор строит и анализирует нелинейную модель системы с управлением с запаздыванием. Стоит отметить, что, ссылаясь работу [9], С.С. Охотников говорит о том, что при внедрении полученного решения необходима полная замена программного обеспечения всех занимаются обслуживанием межсетевых каналов маршрутизаторов. В связи с этим, в своей работе С.С. Охотников предлагает не замену алгоритма RED, а дополнение к этому механизму, такое как динамический корректор, который позволит расширить область устойчивости системы. Из полученных результатов делаются выводы о том, при малых задержках лучшее управление обеспечивает ПИ-регулятор, при больших же задержках наилучшие результаты показывает алгоритм RED с внедренным динамическим корректором.

В статье [14] авторами были разработаны дискретная имитационная модель и непрерывная аналитическая модель системы с TCP-источником и модулем управления RED. Для моделирования источника TCP, модуля управления RED и процесса их взаимодействия авторы предлагают использовать гибридный

(непрерывно-дискретный) подход, который позволяет учитывать переходы между различными состояниями в непрерывной модели протокола TCP и упрощает рассмотрение модели за счет преобразования дифференциальных включений в систему дифференциальных уравнений с дискретными переходами.

## 1.2 Гибридный подход к моделированию систем с управлением

Прежде чем приступать к моделированию сложных систем, необходимо выбрать правильный и корректный модельный подход. При функциональном подходе система рассматривается как выполняющее определенные функции единое целое. При структурном модельном подходе особое внимание уделяется внутренней организации системы со своим устройством управления и правилами функционирования, а также характеру изменения системы в зависимости от воздействия на нее внешних сил.

В статьях [15,16] представлены некоторые методы моделирования. Среди возможных методов исследования выделяют построение дискретно – событийной модели, построение непрерывной модели, а также гибридное моделирование, сочетающее в себе как возможность дискретного, так и непрерывного подхода. В гибридных системах сочетается работа непрерывных и дискретных элементов, например, системы с дискретным устройством управления и объектом управления с непрерывным характером функционирования.

В данной работе рассматривается гибридный подход к моделированию системы, работающей по протоколу TCP с алгоритмом управления RED. При моделировании TCP-подобного трафика можно воспользоваться жидкостным (непрерывным) подходом, однако особенно важно учитывать дискретные переходы между TCP состояниями и функцию сброса пакетов в алгоритмах типа RED, следовательно, гибридный подход отразит эти важные особенности моделируемой системы.

В качестве инструмента для гибридного моделирования в данной работе выступает среда OpenModelica. В работах [17, 18] приведена классификация средств для моделирования, проведен сравнительный анализ языка Modelica и других средств для гибридного моделирования сложных систем, показаны некоторые преимущества и недостатки данного языка.

# 2. Нелинейная система с управлением

## 2.1 Описание процесса передачи трафика с управлением динамической интенсивностью потока по алгоритму типа RED

TCP на сегодняшний день является одним из основных протоколов управления передачи данных в современных пакетных сетях. Механизм TCP функционирует как транспортный протокол с установлением логического соединения, осуществляет управление потоком данных и исправлением ошибок в сети, используя метод передачи с применением окон. TCP отправляет повторный запрос в случае потери данных и, в случае получения двух копий одного пакета, устраняет дублирование, благородя чему обеспечивает целостность передаваемых данных и уведомляет отправителя о результатах передачи. В 1980 году Ван Якобсон (V. Jacobson) в своей работе [19] реализовал алгоритм контроля и управления перегрузками, регулирующий интенсивность передаваемого потока. Этот процесс состоит из 4 этапов: медленный старт, предотвращение перегрузки, затем быстрая передача и восстановление [19-21]. Стоит отметить, что в разных версиях TCP могут быть использованы не все этапы, либо использованы дополнительные режимы.

Также в сетях передачи данных для контроля потока используется расширение Internet Protocol (IP протокола) Explicit Congestion Notification (ECN) [22], описанное в RFC 3168 [23]. Не отбрасывая пакеты, механизм ECN сообщает как источнику, так и получателю данных о возникновении перегрузки на пути к заданному хосту или сети. Пакеты, вызвавшие затор, маркируются, а отправителю посылается информация о том, что необходимо снизить интенсивность предложенной нагрузки. Использование механизма ECN в сочетании с правильно подобранной политикой активного управления очередью может быть очень эффективным для контроля трафика в сети.

Алгоритм активного управления очередью с алгоритмом управления типа RED используется для контроля и предотвращения перегрузок в очередях маршрутизаторов. Классический алгоритм RED описан в статье [6]. Алгоритмы управления состоянием трафика могут быть представлены как модули управления в сетевом оборудовании, например в маршрутизаторах. Алгоритм RED был разработан для контроля трафика в сети и предотвращения перегрузок в случае их

возникновения. При обнаружении затора алгоритмы типа RED начинают случайным образом отбрасывать пакеты прежде, чем весь допустимый размер очереди полностью заполнится. В процессе отбрасывания пакетов RED уведомляет источник о перегрузке и, тем самым, заставляет TCP подобные протоколы снижать скорость передачи, избегая повторной синхронизации. Для контроля длины очереди и обнаружения момента, когда стоит начинать выборочную потерю пакетов, RED отслеживает текущий размер очереди и сравнивает его с некоторыми заданными в настройках маршрутизатора пороговыми значениями. В системах с управлением по алгоритму RED протокол TCP достаточно быстро находит подходящую скорость передачи информации, а так же удерживает размер очереди и время задержки передачи пакетов на уровне, требуемом для предоставления достойного качества услуг связи [24].

Вероятность сброса или навешивания на пакет маркера сброса зависит от пороговых значений очереди маршрутизатора, учитываются минимальное и максимальное значения. Максимальное пороговое значение – это значение, ниже которого алгоритм RED будет пытаться удержать размер очереди, а минимальный порог – это значение, при превышении которого алгоритм начнет выборочно отбрасывать пакеты, избегая тем самым перегрузки. Говоря о подборе пороговых значений, стоит сказать, что в случае, если минимальный предел окажется слишком маленьким, это может привести к падению пропускной способности сети, а если слишком большим - к увеличению времени пребывания пакетов в очереди, что негативно отразится на качестве обслуживания.

Преимуществом алгоритма RED является не только его эффективность, но и относительно простая реализация на сетевом оборудовании, однако при некоторых начальных значениях параметров в настройках маршрутизаторов в системе возникают автоколебания основных параметров. Такой устойчивый автоколебательный режим работы системы отрицательным образом сказывается на показателях качества обслуживания сети, на пропускной способности и латентности.

## 2.2 Математическая модель системы с управлением по алгоритму типа RED

В работе [7] с помощью стохастических дифференциальных уравнений авторами построена модель взаимодействия входящего TCP-потока и маршрутизатора, обрабатывающего трафик по алгоритму управления типа RED. При анализе и моделировании полученной модели, авторы пришли к выводу, что она работоспособна и применима к системам с большим потоком.

Авторами статьи [25] данная модель была расширена: получено математическое описание динамических переменных в виде автономной системы трёх дифференциальных уравнений, зависящих от параметров алгоритма активного управления очередями типа RED.

Автономная система, полученная в статье [25] математически описывает моделируемую систему следующим образом:

$$\begin{cases} \frac{dE[w(t)]}{dt} = \frac{I(w_{max} - E[w(t)])}{E[T(t)]} + I(w(t)-1)\left[\left(-\frac{E[w(t)]}{2}\cdot\right.\right. \\ \quad \cdot \left(1 - E\left[P_{TO}\left(w(t-\tau)\right)\right]\right)\frac{E[w(t-\tau)]}{E[T(t-\tau)]}\cdot \\ \quad \cdot \left(E\left[p_m\left(\hat{q}(t-\tau)\right)\right] + E\left[p_d\left(\hat{q}(t-\tau)\right)\right]\right) + \\ \quad \left. + (1 - E[w(t)]E[P_{TO}\left(w(t-\tau)\right)]\frac{E[w(t-\tau)]}{E[T(t-\tau)]}E[p_d\left(\hat{q}(t-\tau)\right)]\right]; \\ \frac{dE[q(t)]}{dt} = I(R - E[q(t)])\frac{NE[w(t)]}{E[T(t)]}\left(1 - E\left[p_d\left(\hat{q}(t)\right)\right]\right) - E[C(t)]; \\ \frac{dE[\hat{q}(t)]}{dt} = \frac{\ln(1-w_q)}{\sigma}E[\hat{q}(t)] - \frac{\ln(1-w_q)}{\sigma}E[q(t)]. \end{cases} \quad (2.1)$$

Данная система уравнений определяет поведение следующих параметров системы:

$w(t)$ - средний размер TCP окна (в пакетах),

$q(t)$ - средний размер очереди (в пакетах),

$\hat{q}(t)$ - экспоненциально взвешенное скользящее среднее значение длины очереди.

В данной системе учитываются такие параметры, как $N$ – число поступающих в очередь потоков, $p_m\left(\hat{q}(t)\right)$ – вероятность того, что на пакет будет навешана маркировка на сброс, $p_d\left(\hat{q}(t)\right)$ – функция сброса пакета, $P_{TO}\left(w(t-\tau)\right)$ – функция

сброса пакетов по тайм-ауту, $\sigma$ – время между поступлениями соседних пакетов в систему.

Время $T(t)$ можно рассчитать следующим образом

$$T(t) = \begin{cases} T_b + \dfrac{q(t)}{C(t)}, & q(t) > 0 \\ T_b \end{cases}$$

$$C(t) = \begin{cases} C, & q(t) > C \\ q(t), & q(t) \le C \end{cases}$$

где $T_b$ – время приема-передачи одного пакета (Round Trip Time, сек), $q(t)$ – значение мгновенного размера очереди в момент времени $t$, $C(t)$ – интенсивность обслуженной нагрузки.

Весовой коэффициент экспоненциально взвешенного скользящего среднего размера длины очереди $w_q$ рассчитывается как

$$w_q = 1 - e^{-1/C}.$$

При некотором упрощении вышеописанной математической модели процесса передачи трафика с управлением динамической интенсивностью потока по алгоритму типа RED описать поведение основных функций можно следующим образом:

$$\dot{w}(t) = \frac{1}{T(q,t)}\vartheta(w_{max} - w) - \frac{w(t)w\big(t - T(q,t)\big)}{2T\big(t - T(q,t)\big)}p\big(t - T(q,t)\big); \qquad (2.2)$$

$$\dot{q}(t) = \begin{cases} \big(1 - p(t)\big)\dfrac{w(t)}{T(q,t)}N(t) - C, & q(t) > 0 \\ max\left[\big(1 - p(t)\big)\dfrac{w(t)}{T(q,t)}N(t) - C, 0\right], & q(t) = 0 \end{cases}; \qquad (2.3)$$

$$\dot{\hat{q}}(t) = -\omega_q C\hat{q}(t) + \omega_q C q(t) \qquad (2.4)$$

Рассматриваемая в данной работе система содержит $C$ приборов, что соответствует скорости обработки пакетов в очереди ($C$ пакетов в единицу времени). Также в системе учитываются параметры $N(t)$ - число TCP-сессий, $T(q,t)$ – время приема-передачи (Round Trip Time, RTT, сек).

Уравнение (2.2) описывает динамическое изменение среднего размера TCP окна $w(t)$. Элемент $1/T(q,t)$ отражает фазу медленного старта, во время которой размер окна увеличивается за единицу времени двойного оборота. Компонент $\vartheta(w_{max} - w)$, являющийся функцией Хэвисайда, ограничивает роста TCP-окна.

Функция $T(q,t)$ может быть представлена в виде

$$T(q,t) = \frac{q(t)}{C} + \tau_{p,}$$

где $\tau_p$ - задержка распространения пакета по сети (сек).

Второе слагаемое уравнения (2.2) отражает мультипликативное уменьшение размера окна при каждом обнаружении потери пакета в протоколе TCP.

Уравнение (2.3) моделирует поведение среднего размера очереди $q(t)$. Компонента

$$\big(1 - p(t)\big)\frac{w(t)}{T(q,t)}N(t)$$

отражает увеличение длины очереди при поступлении пакетов, что соответствует средней интенсивности поступления пакетов. Уменьшение длины очереди моделируется разностью средней интенсивности поступления пакетов и пропускной способности $C$.

Уравнение (2.4) отражает изменение функции экспоненциально взвешенного скользящего среднего значения длины очереди $\hat{q}(t)$, которое вводится для некоторого сглаживания выбросов мгновенного значения размера очереди $q(t)$, функционируя как фильтр низких частот. Как только значение $\hat{q}(t)$ превышает некое заданное пороговое значение, маршрутизатор узнает о том, что в системе началась перегрузка и сообщает о этом источнику.

Непосредственно за управление по алгоритму типа RED отвечает вероятностная функция сброса пакета $p(\hat{q})$, значения которой лежат в интервале [0,1]. В статьях [6,7] получена формула для вычисления вероятности маркировки на отбрасывание пакетов:

$$p(\hat{q}) = \begin{cases} 0, & 0 \le \hat{q} < q_{min}, \\ \dfrac{\hat{q} - q_{min}}{q_{max} - q_{min}} p_{max}, & q_{min} \le \hat{q} \le q_{max,} \\ 1, & \hat{q} > q_{max} \end{cases} \qquad (2.5)$$

Данная функция зависит от $\hat{q}(t)$, минимального $q_{min}$ и максимального $q_{max}$ уровня размера очереди, а также от параметра $p_{max}$, задающего часть пакетов, которые будут отброшены в случае, если $\hat{q}(t)$ достигает максимума.

На рисунке 2.1 представлен вид вероятностной функции сброса пакетов в алгоритме RED.

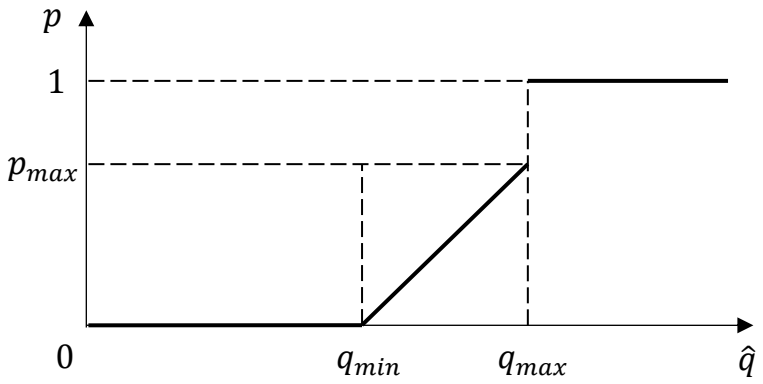


Рисунок 2.1. Вид функции сброса в алгоритме RED

## 2.3 Модификации алгоритма RED

Существует большое количество вариаций классического алгоритма RED, улучшающие те или иные характеристики алгоритма. Для достижения определенных характеристик системы модификации алгоритма могут комбинироваться [26]. В работе [27] представлены математические описания некоторых модификаций алгоритма, проводится классификация этих алгоритмов, также в данной работе выводятся и объясняются параметры и принципы, опираясь на которые была проведена классификация модификаций алгоритма RED.

### *2.3.1 Алгоритм Enhanced Random Early Detection*

Алгоритм с управлением типа Enhanced RED (ERED), проанализированный в статье [27] и представленный в работе [28], отличается от алгоритма RED функцией, описывающей экспоненциально взвешенное скользящее среднее длины очереди. Математическая модель процесса передачи трафика алгоритмом управления ERED может быть представлена следующим образом:

$$\dot{w}(t) = \frac{1}{T(q,t)} - \frac{w(t)w\big(t - T(q,t)\big)}{2T\big(t - T(q,t)\big)} p\big(t - T(q,t)\big);$$

$$\dot{q}(t) = \begin{cases} \big(1 - p(t)\big)\frac{w(t)}{T(q,t)} N(t) - C, & q(t) > 0 \\ max\left[\big(1 - p(t)\big)\frac{w(t)}{T(q,t)} N(t) - C, 0\right], & q(t) = 0 \end{cases} \quad (2.6)$$

$$\hat{q}(t) = \begin{cases} \frac{1-\omega_q}{\alpha}\hat{q}(t) + \omega_q q(t), & q(t) < q_{min}, \alpha > 1 \\ \big(1 - \omega_q\big)\hat{q}(t) + \omega_q q(t), & q_{min} \le q < q_{max} \\ \big(1 - \omega_q\big)\hat{q}(t) + \frac{\omega_q q(t)}{\beta}, & q \ge q_{max}, \beta > 1 \end{cases}$$

где $\alpha$ и $\beta$ — это коэффициенты, управляющие задержкой реагирования $\hat{q}(t)$ на изменение функции $q(t)$. Благодаря данному механизму в системе под управлением алгоритмом ERED амплитуда и частота колебаний экспоненциально взвешенной скользящей средней длины очереди уменьшаются (рис. 2.2).

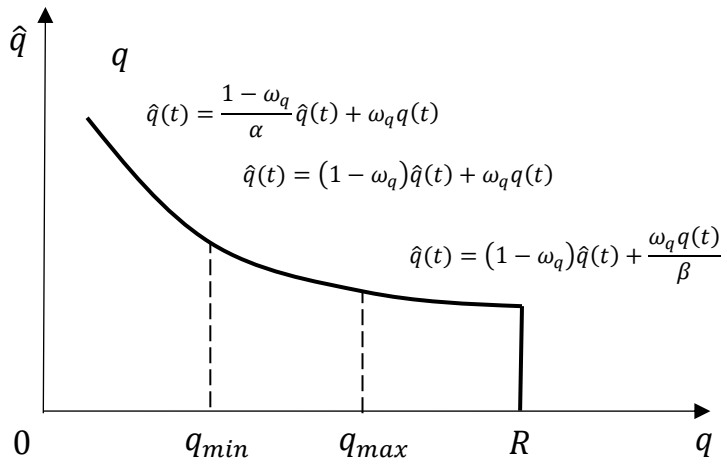


Рисунок 2.2. Функция экспоненциально взвешенного скользящего среднего значения длины очереди в алгоритме ERED

### *2.3.2 Алгоритм Double Slope Random Early Detection*

Математическая модель системы с управлением типа Double Slope RED (DSRED) [29] соответствует уравнениям (2.2 - 2.4). Для вычисления функции сброса пакетов вводится дополнительное пороговое значение $q_{mid}$, оно задается как $q_{mid} = 0{,}5\ (q_{max} - q_{min})$.

Уравнение функции сброса пакетов в алгоритме DSRED, имеет вид:

$$p(\hat{q}) = \begin{cases} 0, & \hat{q} < q_{min}, \\ \alpha(\hat{q} - q_{min}), & q_{min} \le \hat{q} < q_{mid}, \\ 1 - \gamma + \beta(\hat{q} - q_{mid}), & q_{mid} \le \hat{q} < q_{max}, \\ 1, & \hat{q} \ge q_{max}, \end{cases} \quad (2.7)$$

где $\alpha = \frac{2(1-\gamma)}{q_{max}-q_{min}}$, $\beta = \frac{2\gamma}{q_{max}-q_{min}}$.

Вероятностная функция сброса пакетов имеет вид аппроксимированной ломаной линии с двумя звеньями. Коэффициент $\alpha$ задает наклон функции на первом промежутке от $q_{min}$ до $q_{mid}$, на данном участке перегрузка в сети еще не критична, поэтому $\alpha$ выбирается таким образом, чтобы отбросить как можно меньше пакетов. За наклон функции на втором сегменте отвечает переменная $\beta$. На данном участке перегрузка в сети должна быть устранена как можно быстрее, значит, чтобы оповестить источник о перегрузке, вероятность сброса пакетов должна быть достаточно высокой, следовательно, $\alpha < \beta$ [27].

Вид вероятностной функции сброса в алгоритме DSRED представлен на рисунке 2.3.

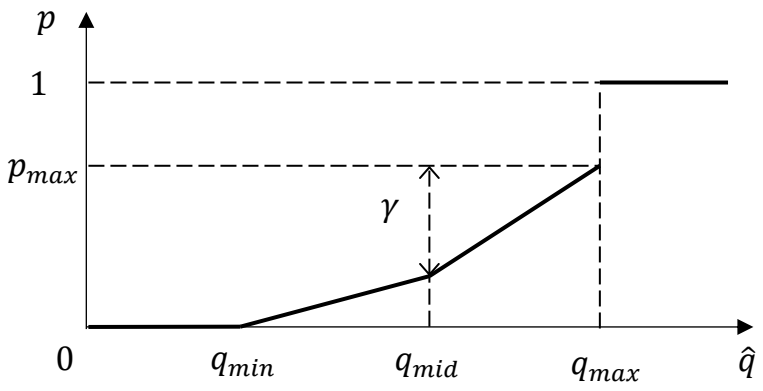


Рисунок 2.3. Вид функции сброса в алгоритме DSRED

### *2.3.3 Алгоритм Gentle Random Early Detection*

Функция вероятности сброса пакетов в алгоритме Gentle RED (GRED) [30] имеет вид:

$$p(\hat{q}) = \begin{cases} 0, & \hat{q} < q_{min}, \\ \dfrac{\hat{q} - q_{min}}{q_{max} - q_{min}} p_{max}, & q_{min} \le \hat{q} < q_{max}, \\ \dfrac{\hat{q} - q_{max}}{q_{max}} (1 - p_{max}) + p_{max}, & q_{max} \le \hat{q} < 2q_{max}, \\ 1, & \hat{q} \ge 2q_{max} \end{cases} \quad (2.8)$$

В системе с управлением по GRED в диапазоне от $q_{min}$ до $q_{max}$ находится стандартная функция сброса пакетов классического RED, представленная в уравнении (2.5). Дополнительный сегмент от $q_{max}$ до $2q_{max}$ вводится для того, чтобы сделать поведение функции сброса более плавным на уровне выше максимального порога размера очереди, то есть вне рабочего диапазона классического алгоритма RED (рис. 2.4).

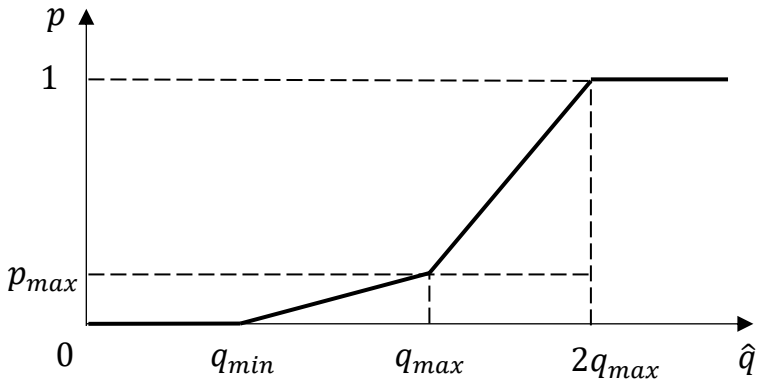


Рисунок 2.4. Вид функции сброса в алгоритме GRED

### *2.3.4 Алгоритм State Dependent Random Early Detection*

Алгоритм State Dependent RED (SDRED) отличается от классического RED функцией расчета $\hat{q}(t)$ и функцией сброса пактов $p(\hat{q})$. Функция вероятности сброса пакетов в данном алгоритме вычисляется следующим образом [31]:

$$p(\hat{q}) = \begin{cases} 0, & \hat{q} < q_{min}, \\ \dfrac{\hat{q} - q_{min}}{q_{max} - q_{min}} p_{max}, & q_{min} \le \hat{q} < q_{max}, \\ \dfrac{\hat{q} - q_{min}}{(q_{max} + 0{,}1R) - q_{min}} p_{max}, & q_{max} \le \hat{q} < 0{,}7R, \\ \dfrac{\hat{q} - q_{min}}{(q_{max} + 0{,}2R) - q_{min}} p_{max}, & 0{,}7R \le \hat{q} < 0{,}8R, \\ \dfrac{\hat{q} - q_{min}}{(q_{max} + 0{,}3R) - q_{min}} p_{max}, & 0{,}8R \le \hat{q} < 0{,}9R, \\ 1, & 0{,}9R \le \hat{q} \le R. \end{cases} \quad (2.9)$$

Функция сброса в алгоритме SDRED продемонстрирована на рисунке 2.5.

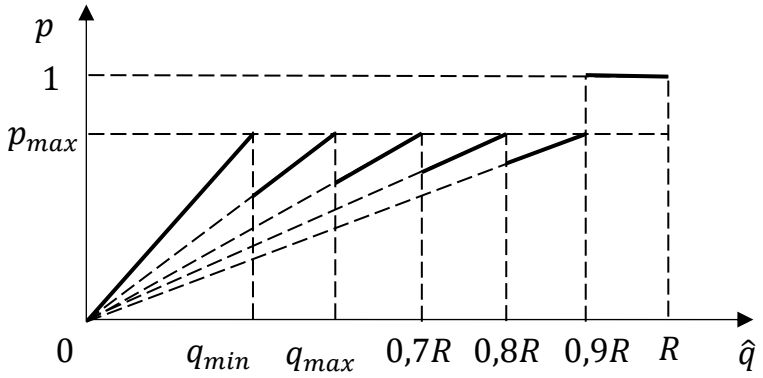


Рисунок 2.5. Вид функции сброса в алгоритме SDRED

Значение экспоненциально взвешенной скользящей средней длины очереди в алгоритме SDRED имеет вид:

$$\hat{q}(t) = \begin{cases} (1 - \omega_q)\hat{q}(t) + \omega_q q(t), & q(t) < q_{max}, 0{,}9 \le q(t) \le R \\ (1 - k\omega_q)\hat{q}(t) + k\omega_q q(t), & q_{max} \le q(t) < 0{,}7R \\ (1 - k^2\omega_q)\hat{q}(t) + k^2\omega_q q(t), & 0{,}7R \le q < 0{,}8R \\ (1 - k^3\omega_q)\hat{q}(t) + k^3\omega_q q(t), & 0{,}8R \le q < 0{,}9R \end{cases} \quad (2.10)$$

где $k$ – константа для изменения $w(t)$ в зависимости от коэффициента, отражающего занятость очереди.

# 3. Моделирование системы с управлением в среде OpenModelica

При моделировании таких нелинейных систем, как системы с управлением типа RED, оптимально использовать гибридный подход, реализованный с помощью средства OpenModelica. OpenModelica— ориентированная для промышленного и академического применения среда для моделирования, опирающаяся на открытый код и язык Modelica [32].

Modelica - объектно-ориентированный моделирования, разработанный некоммерческой организацией Modelica Association. Modelica хорошо применима для компонентно-ориентированного моделирования сложных систем, состоящих из различных физических компонентов, также имеющих компоненты управления и элементы, ориентированные на отдельные процессы.

## 3.1 Численное моделирование системы с управлением

В качестве численно моделируемой системы с управлением выступает система, функционирующая по алгоритму RED. Полная реализация вычислительной модели системы с управлением по алгоритму типа RED и его модификациям на языке Modelica представлена в разделе Приложение.

### *3.1.1 Численное моделированием алгоритма RED*

Алгоритм, отражающий изменение размера TCP-окна (2.2) представлен ниже (Листинг 3.1).

Листинг 3.1. Уравнение для изменения размера окна в RED

```
equation
der(w) = wAdd ( w, wmax, T ) - w * delay ( w, T )  * delay ( p,
T ) / ( 2 * delay ( T, T ) );
when w <= 1.0
    then reinit ( w, 1.0 ) ;
end when;
when q >= R
    then reinit ( q, R ) ;
end when;
```

На языке Modelica `delay` задает запаздывание, а оператор `der` - производную по времени. Функция `wAdd` необходима для ограничения роста размера TCP-окна (Листинг 3.2).

Листинг 3.2. Функция ограничения роста окна в RED

```
function wAdd
  input Real wIn;
  input Real wmax;
  input Real T;
  output Real wOut;
algorithm
  wOut := if noEvent( wIn >= wmax ) then 0 else 1 / T;
end wAdd;
```

Ниже представлен фрагмент кода, задающий уравнение изменения средней длины очереди (Листинг 3.3).

Листинг 3.3. Уравнение для изменения длины очереди в RED

```
equation
der(q) = qAdd(pre(q),w,T,C,N,R);
```

Функция изменения среднего размера очереди, реализованная на языке Modelica, представлена ниже (Листинг 3.4).

Листинг 3.4. Функция изменения длины очереди в RED

```
function qAdd
  input Real q;
  input Real w;
  input Real T;
  input Real C;
  input Real N;
  input Real R;
  output Real qOut;
protected
  Real q1;
  Real q2;
algorithm
  q1 := N * w / T - C;
  q2 := q + q1;
  qOut := if ( q2 > R ) then R - q elseif ( q2 > 0 ) then q1
else - q;
end qAdd;
```

Согласно уравнению 2.4, задаем экспоненциально взвешенное скользящее среднее значение длины очереди (Листинг 3.5).

Листинг 3.5. Уравнение взвешенного скользящего среднего длины очереди в RED

```
equation
der(q_avg) = wq * C * ( q - q_avg );
```

Листинг 3.6 на языке Modelica отражает функцию сброса в классическом алгоритме RED. В гибридном подходе кусочная и разрывная функция сброса задаётся единым оператором.

Листинг 3.6. Уравнение функции сброса пакетов в RED

```
equation
p = if ( q_avg < thmin * R ) then 0.0 elseif ( q_avg > thmax *
R ) then 1.0 else ( q_avg / R – thmin ) * pmax / ( thmax –
thmin );
```

### *3.1.2 Численное моделированием систем с модификациями алгоритма RED*

Проведем математическое моделирование систем, работающих по модификациям алгоритма RED.

#### *Алгоритм ERED*

Листинг 3.7 демонстрирует реализацию функции $\hat{q}(t)$, отличающей алгоритм ERED от классического RED.

Листинг 3.7. Уравнение взвешенного скользящего среднего длины очереди в ERED

```
equation
der(q_avg) = if q < thmin * R and alfa > 1 then wq * C * ( q -
q_avg / alfa )
elseif q >= thmax * R and betta > 1 then wq * C * ( q / betta -
q_avg)
else wq * C * ( q - q_avg );
```

#### *Алгоритм DSRED*

Алгоритм DSRED отличается от классического RED функцией сброса пакетов (уравнение 2.7). Листинг 3.8 отражает реализацию функции $p(t)$ в данном алгоритме на языке Modelica.

Листинг 3.8. Уравнение функции сброса пакетов в DSRED

```
equation
p = if q_avg < thmin * R then 0.0
elseif q_avg >= thmin * R and q_avg < thmid * R then alfa * (
q_avg - thmin * R)
elseif q_avg >= thmid * R and q_avg < thmax * R then 1 - gamma
+ betta * ( q_avg - thmin * R )
else 1;
```

### *Алгоритм GRED*

Ввод дополнительного сегмента в функцию сброса пакетов для более плавного поведения системы вне рабочего диапазона классического алгоритма RED (уравнение 2.8) представлен ниже (Листинг 3.9).

Листинг 3.9. Уравнение функции сброса пакетов в GRED

```
equation
p = if q_avg < thmin * R then 0.0
elseif q_avg >= thmin * R and q_avg < thmax * R then ( (q_avg -
thmin * R) * pmax ) / ( thmax * R - thmin * R )
elseif q_avg >= thmax * R and q_avg < 2 * thmax * R then ( (
q_avg - thmax * R ) * ( 1 - pmax ) ) / ( thmax * R) + pmax
else 1.0;
```

### *Алгоритм SDRED*

Фрагмент кода, задающий уравнение экспоненциально взвешенной скользящей средней длины очереди в алгоритме SDRED (уравнение 2.9) представлен далее (Листинг 3.10).

Листинг 3.10. Уравнение взвешенного скользящего среднего длины очереди в SDRED

```
equation
der ( q_avg ) = if q >= thmax * R and q < 0.7 * R then wq * C *
k * ( q - q_avg )
elseif q >= 0.7 * R and q < 0.8 * R then wq * C * k^2 * ( q -
q_avg )
elseif q >= 0.8 * R and q < 0.9 * R then wq * C * k^3 * ( q -
q_avg )
else wq * C * ( q - q_avg );
```

Функция сброса пакетов в системе с управлением по алгоритму SDRED так же задается в виде атомарного оператора (Листинг 3.11).

Листинг 3.11. Уравнение функции сброса пакетов в SDRED

```
equation
p = if q_avg < thmin * R then 0.0
elseif q_avg >= thmin * R and q_avg < thmax * R then ( ( q_avg
- thmin * R ) * pmax ) / ( thmax * R - thmin * R )
elseif q_avg >= thmax * R and q_avg < 0.7 * R then ( ( q_avg -
thmin * R ) * pmax ) / ( thmax * R + 0.1 * R - thmin * R )
elseif q_avg >= 0.7 * R and q_avg < 0.8 * R then ( ( q_avg -
thmin * R ) * pmax ) / ( thmax * R + 0.2 * R - thmin * R )
elseif q_avg >= 0.8 * R and q_avg < 0.9 * R then ( ( q_avg -
thmin * R ) * pmax ) / ( thmax * R + 0.3 * R - thmin * R )
else 1.0;
```

## 3.2 Имитационное моделирование системы с управлением

### *3.2.1 Имитационное моделирование гибридной модели протокола TCP в Octave*

Прежде чем приступить к разработке и реализации гибридной модели системы с управлением, работающей по алгоритму RED в среде OpenModelica, рассмотрим и подробно разберем гибридную модель протокола TCP Newreno, предложенную в работе [33], реализованную на языке Matlab. Повторим предложенную в этой работе реализацию имитационной модели в программной среде Octave, исправим ошибки в функции, описывающей смену TCP-состояний, для корректной работы модели подберем значения начальных параметров и констант, при анализе и подборе которых использовалась статья [34], а также усовершенствуем функцию вывода результирующих графиков.

Гибридная модель представлена в виде трех функций, описывающих поведение работающего по протоколу TCP Newreno источника, обработку очередей в сети и вывод результатов моделирования. Полный код программы представлен в разделе Приложение 2.

TCP Newreno [19,20] запускается в режиме медленного старта, где окно перегрузки $w_f$ для потока $f$ удваивается за каждый RTT. Этот экспоненциальный рост моделируется следующим образом:

$$\dot{w_f} = \frac{\log m}{RTT_f} w_f, \tag{3.1}$$

где $m$ – константа, определяющая скорость роста, а $RTT_f$ – время приёма-передачи для потока $f$.

Окно перегрузки источника (Source congestion window, cwnd) увеличивается при получении ACK согласно следующей формуле:

$$cwnd_{(n+1)} = cwnd_{(n)} + 1, \tag{3.2}$$

где $n$ индексирует полученные ACK.

Мгновенная скорость передачи $r_f$ для каждого потока определяется как

$$r_f = \frac{\beta w_f}{RTT_f}, \tag{3.3}$$

где $\beta$ отражает изменения, связанные с быстрым изменением RTT, наблюдаемым при медленном старте.

При превышении размера TCP окна порога медленного старта (Slow start threshold, ssthr) TCP выходит из режима медленного запуска. Если же размер TCP окна не достигает данного порога, то $w_f$ увеличивается то тех пор, пока пришедший в систему пакет не будет отправлен в очередь. Чтобы отразить то, что существует задержка между моментом, когда пакет отбрасывается, и моментом, когда TCP-источник обнаруживает это отбрасывание, существует режим задержки медленного старта, где переменные $w_f$ и $r_f$ продолжают увеличиваться согласно уравнениям 3.1, 3.3 в течении времени задержки обнаружения потери (drop detection delay, $DDD_f$). В зависимости от количества потерянных пакетов $n_{drop}$, система перейдет либо в фазу быстрого восстановления, либо в состояние тайм-аута.

В состоянии быстрого восстановления, переменные $w_f$ и $r_f$ изменяются в соответствии со следующими уравнениями:

$$w_f = \frac{w_{\bar{f}}}{2}, \tag{3.4}$$

$$r_f = \frac{1 + \frac{w_{\bar{f}}}{2} - n_{drop}}{RTT_f}, \tag{3.5}$$

где $w_{\bar{f}}$ - значение $w_f$ в момент, когда источник обнаружил первое отбрасывание пакета.

В режиме предотвращения перегрузки $w_f$ увеличивается линейно со скоростью один пакет на $RTT_f$. В данном состоянии переменные $w_f$ и $r_f$ вычисляются в соответствии со следующими уравнениями:

$$\dot{w}_f = \frac{1}{RTT_f} w_f, \tag{3.6}$$

$$r_f = \frac{w_f}{RTT_f}. \tag{3.7}$$

Окно перегрузки увеличивается в соответствии со следующей формулой:

$$cwnd_{(n+1)} = cwnd_{(n)} + \frac{1}{cwnd_{(n)}}. \tag{3.8}$$

Система переходит в режим ожидания, только из режима задержки медленного запуска или режима задержки перегрузки, если выполняется условие $w_f \leq \max\{2 + n_{drop}, 2n_{drop} - 4\}$.

Модель предполагает, что TCP-источник остается в этом режиме в течение 1 секунды и $r_f = 0$. После этого режима переменная $w_f$ снижается до 1, $ssthr_f$ опускается до значения $w_{\bar{f}}/2$.

На основе описания переходов между фазами протокола TCP Newreno можно построить диаграмму (рис. 3.1) [34].

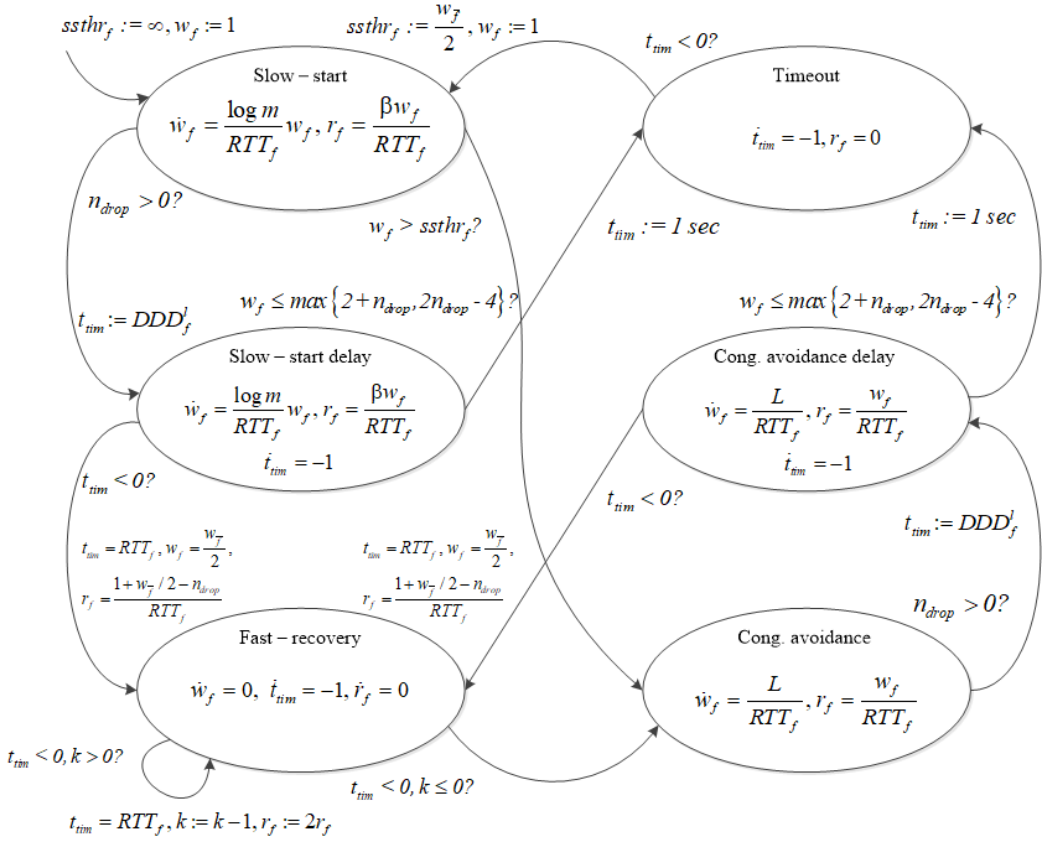


Рисунок 3.1 Диаграмма переходов между состояниями в гибридной модели в протоколе TCP Newreno

Полученную диаграмму можно преобразовать в программу на языке MatLab (Приложение 2).

Для некоторого упрощения модели в качестве политики обработки очередей применяется политика Tail drop. В функции, отвечающей за обработку очередей, выделяются три состояния: «очередь пуста», «очередь не заполнена» и «очередь заполнена».

Пусть $F$ - множество потоков $\{f_1, f_2, \dots, f_n\}$, а $L$ - множество каналов связи $\{l_1, l_2, \dots, l_n\}$. В режиме пустой очереди $s_f^l$ обозначает входящую скорость для потока $f$ по каналу $l$, переменная $r_f^l$ – скорость передачи потока $f$ канала $l$. Размер очереди моделируется как дифференциальное уравнение первого порядка, определяемое следующим образом:

$$\dot{q_f^l} = s_f^l - r_f^l \quad \forall f \in F, l \in L \tag{3.9}$$

$$r_f^l = \begin{cases} s_f^l, & \text{если } s^l \leq B^l \\ \frac{s_f^l}{s^l} B^l, & \text{если } s^l > B^l \end{cases}, \tag{3.10}$$

где $s^l$ – общая входящая скорость, $B^l$ – интенсивность обслуживания очереди.

В случае, когда очередь не заполнена, размер очереди вычисляется по формуле 3.9, а исходящая скорость потока вычисляется следующим образом:

$$r_f^l = \frac{q_f^l}{q^l} B^l, \tag{3.11}$$

где $q^l$ – общий размер очереди.

Если очередь полностью заполнена, то исходящая скорость потока моделируется по формуле 3.11, размер очереди изменяется соответственно уравнению:

$$\dot{q_f^l} = \left(\frac{s_f^l}{s^l} - \frac{q_f^l}{q^l}\right) B^l \tag{3.12}$$

На основе вышеописанного принципа обработки очередей можно построить диаграмму переходов состояний очереди (рис. 3.2).

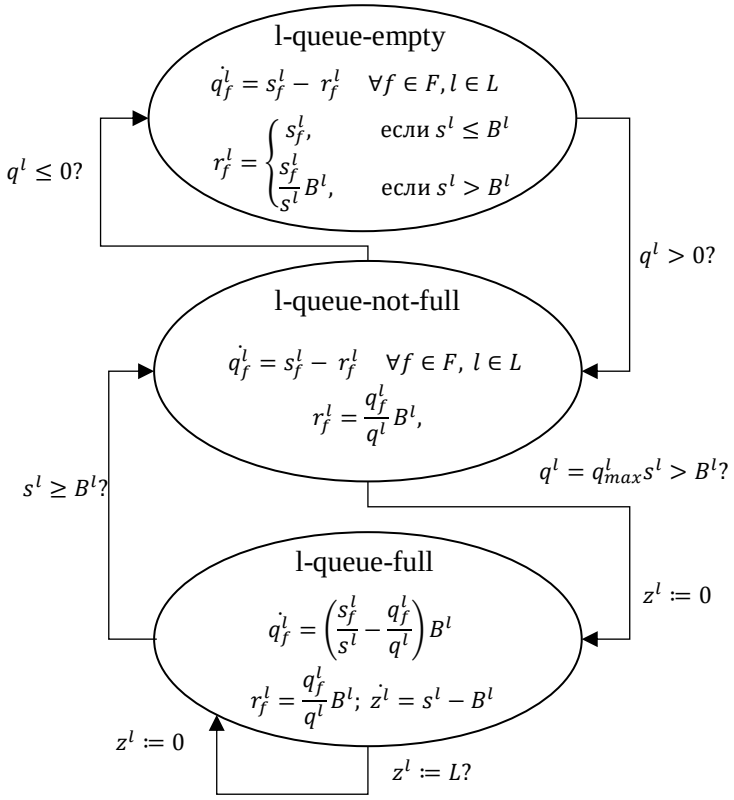


Рисунок 3.2. Диаграмма переходов между состояниями в модели обработки очередей

### *3.2.1 Имитационное моделирование гибридной системы с управлением по алгоритму типа RED в OpenModelica*

Базируясь на вышеописанные принципы функционирования протокола TCP Reno, в среде OpenModelica была построена имитационная модель системы с управлением по алгоритму типа RED.

Гибридная модель представлена в виде классов, соединенных коннекторами и описывающих поведение следующих элементов системы: TCP-источника, получателя, двух маршрутизаторов, каналов передачи данных и обработку очередей в сети в соответствии с RED. Топология описываемой системы представлена на рисунке 3.3.

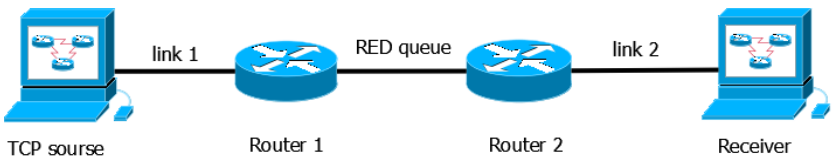


Рисунок 3.3. Топология моделируемой сети

Для некоторого упрощения и уменьшения количества состояний источник функционирует по протоколу TCP Reno [35]. Система начинает работу в состоянии медленного старта. Переменная $ssth$ используется для перехода от фазы медленного старта к фазе предотвращения перегрузки. Когда новое соединение устанавливается, эта переменная возрастает до максимально возможного размера окна. Режим медленного старта продолжается до тех пор, пока значение $cwnd$ не достигнет $ssth$, затем происходит переход к состоянию предотвращения перегрузки. Когда происходит потеря пакета, TCP переходит либо в фазу быстрого восстановления, либо в состояние тайм-аута. Основываясь на описании переходов TCP состояний, можно построить UML-диаграмму, демонстрирующую переходы между состояниями (рис. 3.4).

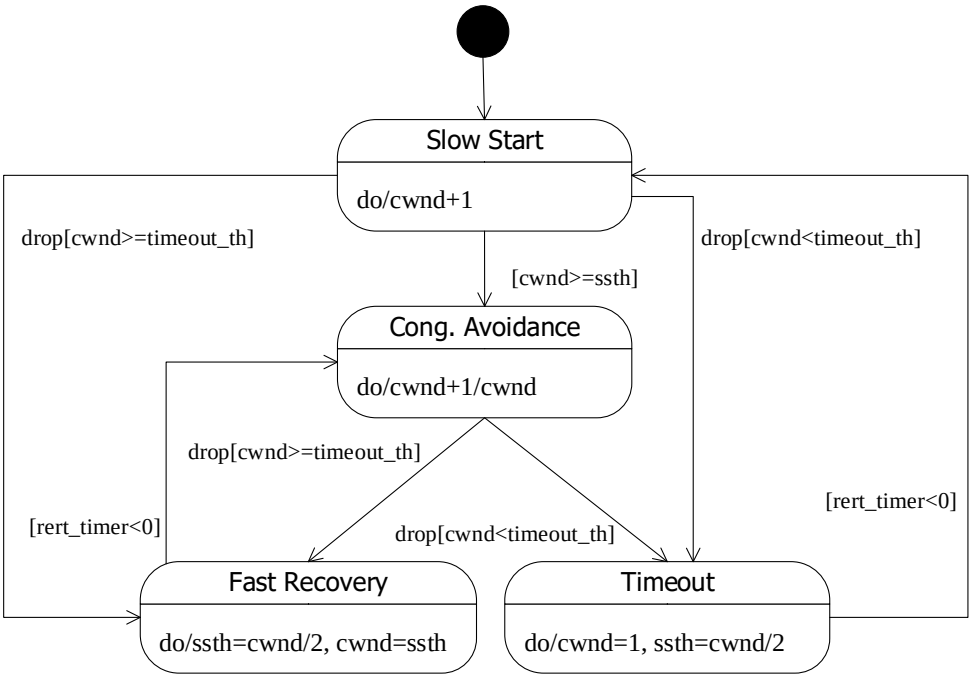


Рисунок 3.4. Диаграмма переходов между состояниями в гибридной модели в протоколе TCP Reno

Данную диаграмму можно преобразовать в код на языке Modelica. Листинг 3.12 отражает алгоритм переходов между состояниями TCP протокола. Полный код программы представлен в разделе Приложение 3.

Листинг 3.12. Алгоритм смены TCP состояний

```
algorithm
state := TCPState_slowStart;
when edge(DDD) and w >= timeout__threshold  and (state ==
TCPState_slowStart or state == TCPState_congestion_Avoid) then
state := TCPState_fastRecov;
```

```
elsewhen w >= ssth and state == TCPState_slowStart then
state := TCPState_congestion_Avoid;
elsewhen w < timeout_th and edge(DDD) and (state ==
TCPState_s_start_mode or state == TCPState_congestion_Avoid)
then
state := TCPState_timeOut;
  elsewhen retr_timer < 0 and state ==
TCPState_fast_recovery_mode then
  state := TCPState_congestion_avoidance;
  elsewhen retr_timer < 0 and state == TCPState_timeOut then
  state := TCPState_s_start_mode;
  end when;
```

Уравнения для описания изменения среднего размера TCP окна отражает Листинг 3.13.

Листинг 3.13. Изменения среднего размера окна в TCP

```
equation
  // Triple Duplicate
  when (pre(state) == TCPState_s_start_mode or pre(state) ==
TCPState_congestion_avoidance) and state ==
TCPState_fast_recovery_mode then
    reinit(retr_timer, o.RTT);
    reinit(ssth, w / 2);
    reinit(w, w / 2);
  end when;
  // Timeout
  when (pre(state) == TCPState_s_start_mode or pre(state) ==
TCPState_congestion_avoidance) and state == TCPState_timeOut
then
    reinit(retr_timer, RTO);
    reinit(ssth, -w / 2);
    reinit(w, 1);
  end when;
  // Equations for the 4 different states
  if state == TCPState_congestion_avoidance then
    // additive increase
    der(w) = a / o.RTT;
    o.rate = w * L / o.RTT;
    der(retr_timer) = 0;
  elseif state == TCPState_fast_recovery_mode then
    der(w) = 0;
    o.rate = w * L / o.RTT;
    der(retr_timer) = -1;
  elseif state == TCPState_timeOut then
    der(w) = 0;
    o.rate = 0.001;
    der(retr_timer) = -1;
  else
    // state == s_start_mode
    // exponential increase
    der(w) = Modelica.Math.log(m) * w / o.RTT;
```

```
  o.rate = w * L / o.RTT;
  der(retr_timer) = 0;
end if;
der(drop_timer) = if drop_timer > 0 then -1.0 else 0.0;
DDD = if drop_timer < 0 then true else false;
when o.drop and drop_timer <= 0 then
  reinit(drop_timer, o.RTT);
end when;
```

## 3.3 Анализ полученных в ходе моделирования результатов

### *3.3.1 Результаты численного моделирования системы с управлением*

В работе [36] предложен способ определения области, на которой в системе передачи TCP трафика возникает автоколебательный режим, предложен метод поиска стационарной точки, а также представлены численные вычисления для алгоритма RED. Используя данные, полученные в вышеописанной статье, продемонстрируем поведение динамических переменных системы (2.2) - (2.6) при разных начальных значениях.

В качестве изменяемых параметров выступают переменные $th_{min}$ и $th_{max}$, нормализованный минимальный и максимальный пороги очереди, параметр $w_q$ и переменная $p_{max}$, показывающая часть пакетов, отбрасываемых в случае обнаружение потери.

Зададим следующие начальные значения параметров: $N = 60{,}0; C = 10{,}0; T = 0{,}5; th_{min} = 0{,}25;\ th_{max} = 0{,}5;\ R = 300{,}0;\ w_q = 0{,}0004;\ p_{max} = 0{,}1;\ w_{max} = 32{,}0.$

В результате моделирования системы с данными параметрами была получена динамика изменения, $w(t), q(t), \hat{q}(t)$ продемонстрированная на рисунке 3.5, и фазовая траектория (рис. 3.6), которая осциллирует вокруг своей стационарной точки. Фазовый портрет в виде предельного цикла говорит о том, что при данных значениях параметров в системе возникает устойчивый автоколебательный режим функционирования. Автоколебания отрицательно сказываются на таких параметрах и показателях сети, как на ее пропускную способность и задержку передачи пакетов.

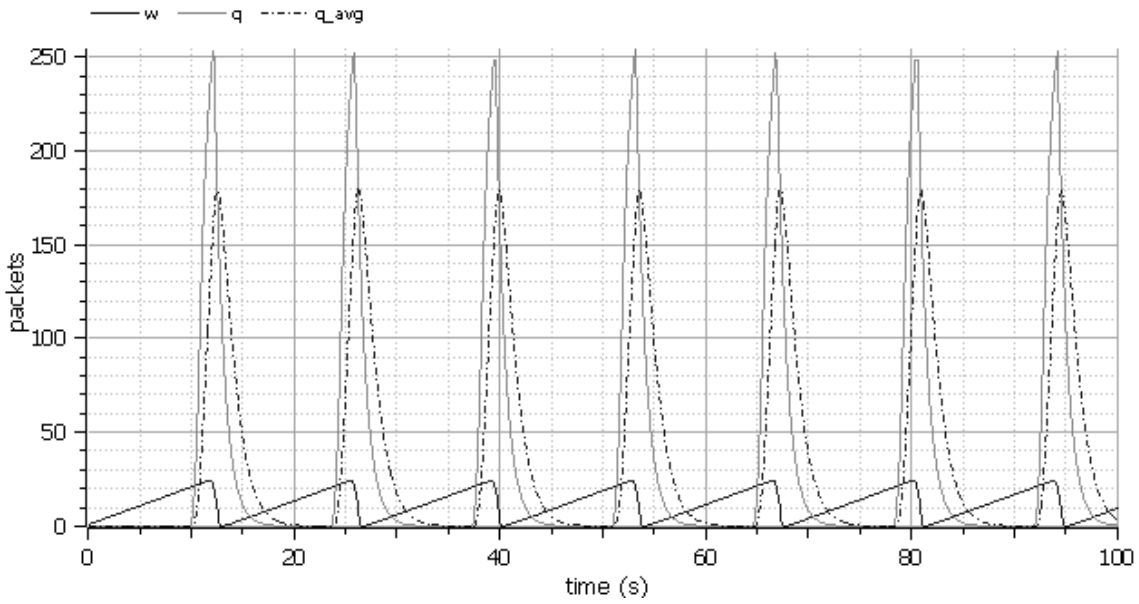


Рисунок 3.5. Динамика изменения $w(t), q(t), \hat{q}(t)$ в алгоритме RED

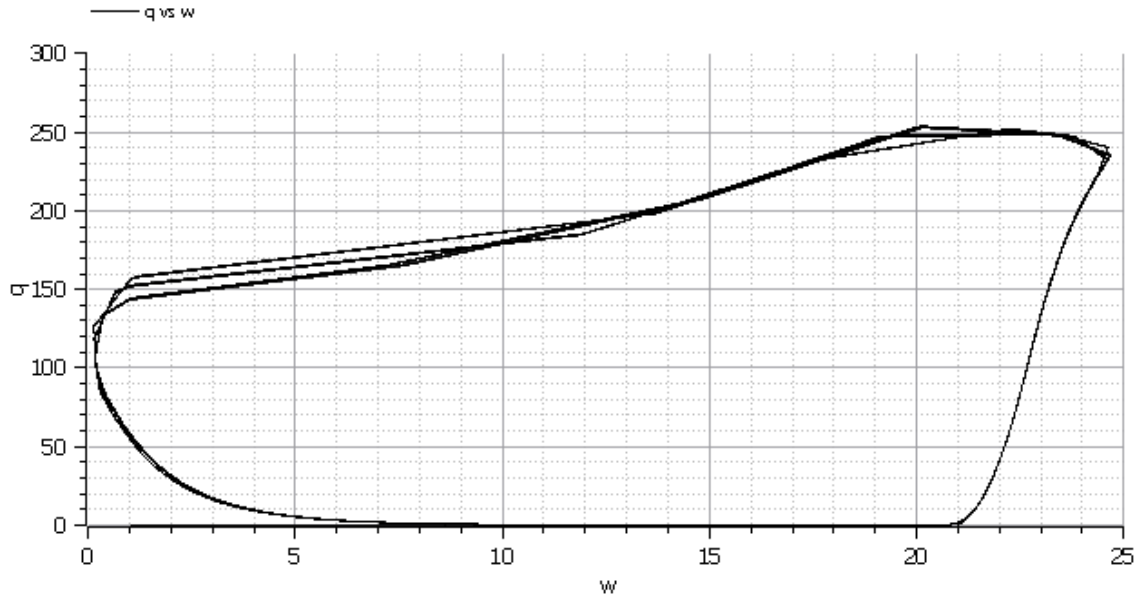


Рисунок 3.6. Фазовый портрет $(w, q)$ в алгоритме RED

В случае изменения начальных значений системы на $N = 60{,}0; C = 10{,}0; T = 0{,}5; th_{min} = 0{,}5;\ th_{max} = 0{,}9;\ R = 300{,}0;\ w_q = 0{,}0001;\ p_{max} = 0{,}01;\ w_{max} = 32{,}0,$ т.е. при увеличении промежутка, на котором действует функция сброса пакетов и при уменьшении доли отбрасываемых в случае перезарузки пакетов, амплитуда колебаний функции размера TCP-окна уменьшилась, а амплитуда среднего размера очереди, наоборот, увеличилась. Однако частота колебаний всех функций $w(t), q(t), \hat{q}(t)$ уменьшилась, что говорит о более устойчивом поведении системы

при данных параметрах. Динамика изменения $w(t), q(t), \hat{q}(t)$ и фазовый портрет $(w, q)$ представлены на рисунках 3.7 и 3.8 соответственно.

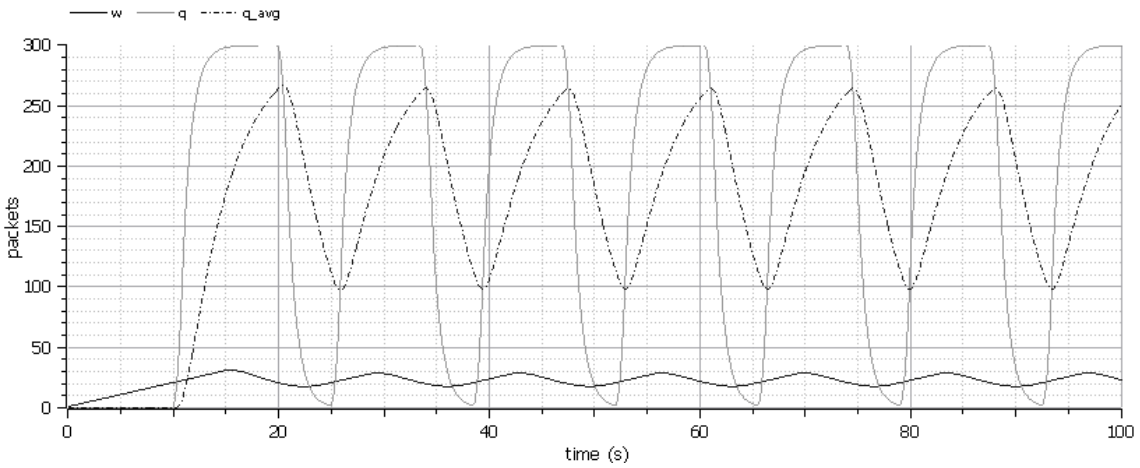


Рисунок 3.7. Динамика изменения $w(t), q(t), \hat{q}(t)$ в алгоритме RED

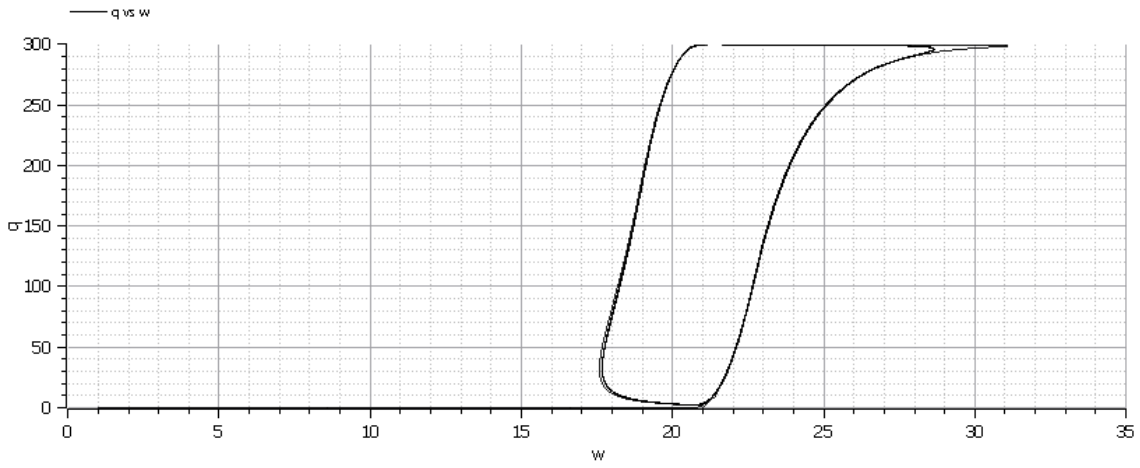


Рисунок 3.8. Фазовый портрет $(w, q)$ в алгоритме RED

Продемонстрируем сходства и различия в поведении систем, функционирующих по RED и его модификациям.

Результаты моделирования системы, работающей по алгоритму ERED, представлены ниже. Начальные параметры: $N = 60{,}0; C = 10{,}0; T = 0{,}5; th_{min} = 0{,}25;\ th_{max} = 0{,}5;\ R = 300{,}0;\ w_q = 0{,}0004;\ p_{max} = 0{,}1;\ w_{max} = 32{,}0.$

Рисунок 3.9 отражает поведение системы при коэффициентах $\alpha = 10{,}0$ и $\beta = 0{,}4$, управляющие задержкой реагирования $\hat{q}(t)$ на функцию длины очереди $q(t)$. При данных коэффициентах частота колебаний функции $\hat{q}(t)$ уменьшается (рис. 3.10), т.е. параметры системы изменяются более плавно, что говорит о более устойчивом характере поведения системы, функционирующей по алгоритму ERED по сравнению с системой по RED.

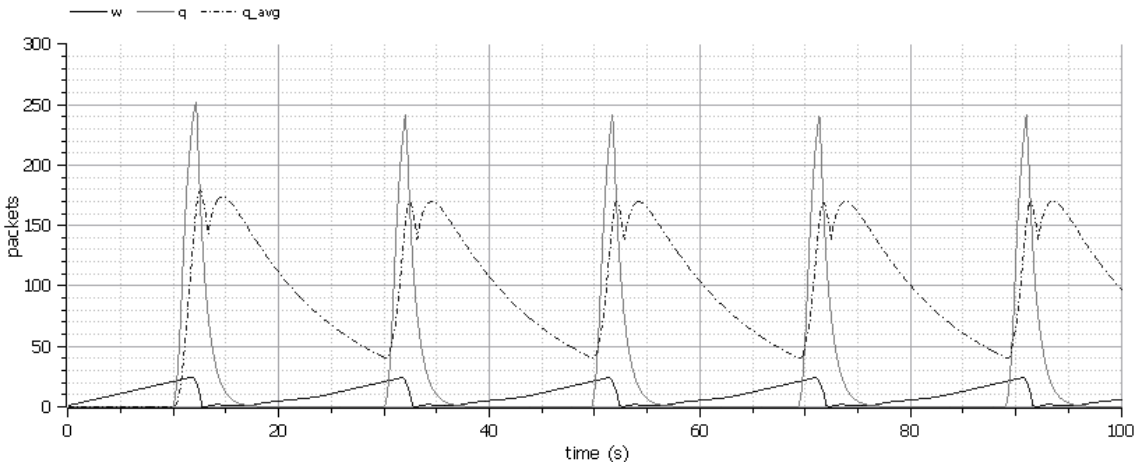


Рисунок 3.9. Динамика изменения $w(t), q(t), \hat{q}(t)$ в системе с управлением типа ERED при $\alpha = 10{,}0$ и $\beta = 0{,}4$

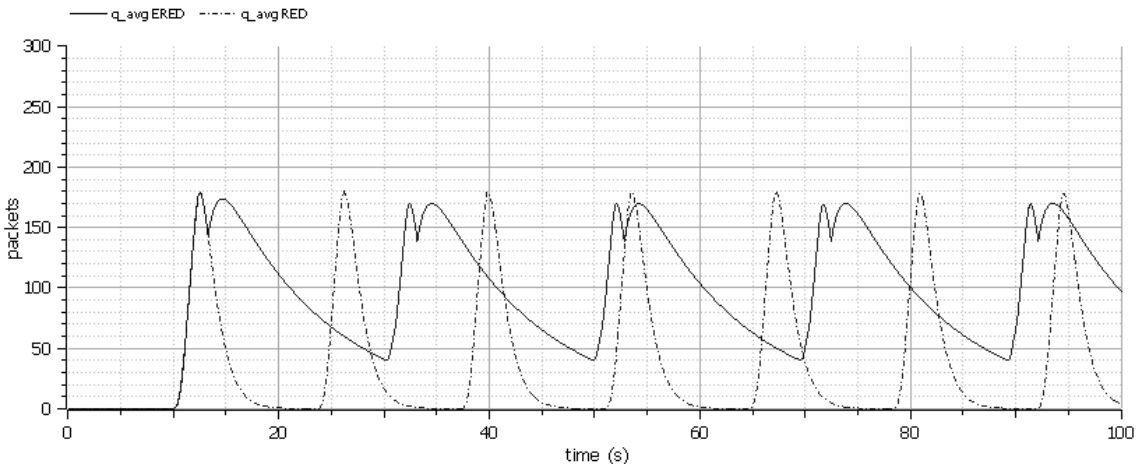


Рисунок 3.10. Различие в поведении переменной $\hat{q}(t)$ в алгоритмах RED и ERED

Изменим параметры системы: $N = 60{,}0; C = 10{,}0; T = 0{,}5; th_{min} = 0{,}5;\ th_{max} = 0{,}9;\ R = 300{,}0;\ w_q = 0{,}0001;\ p_{max} = 0{,}01;\ w_{max} = 32{,}0$.

Продемонстрируем поведение системы при новых параметрах (рис. 3.11) и различие в поведение взвешенной скользящей средней длины очереди в алгоритмах RED и ERED (рис. 3.12). В данном случае при использовании алгоритма ERED также уменьшается частота колебаний экспоненциально взвешенной скользящей средней длины очереди, что снова демонстрирует такое преимущество, как более устойчивый характер работы системы алгоритма ERED по сравнению с RED.

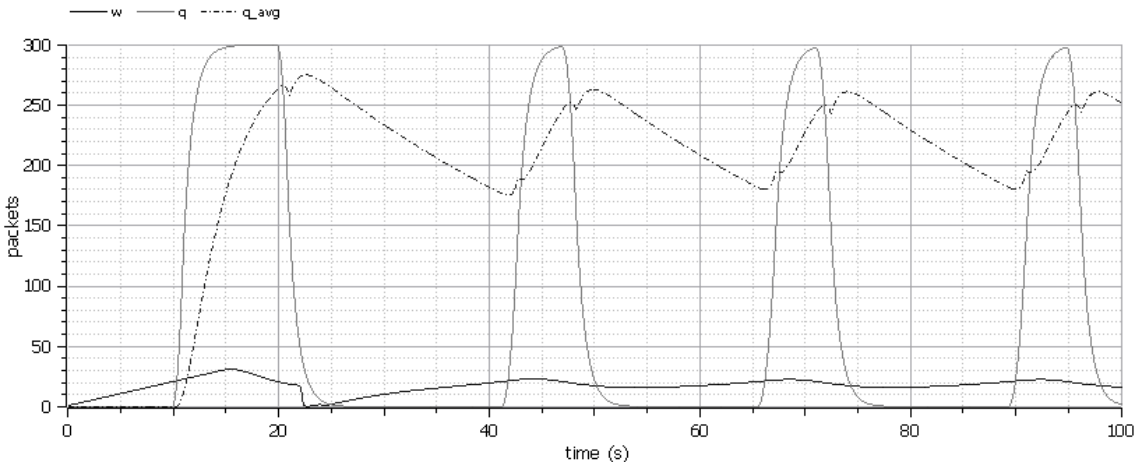


Рисунок 3.11. Динамика изменения $w(t), q(t), \hat{q}(t)$ в системе с управлением типа ERED при $= 10{,}0$ и $\beta = 0{,}4$

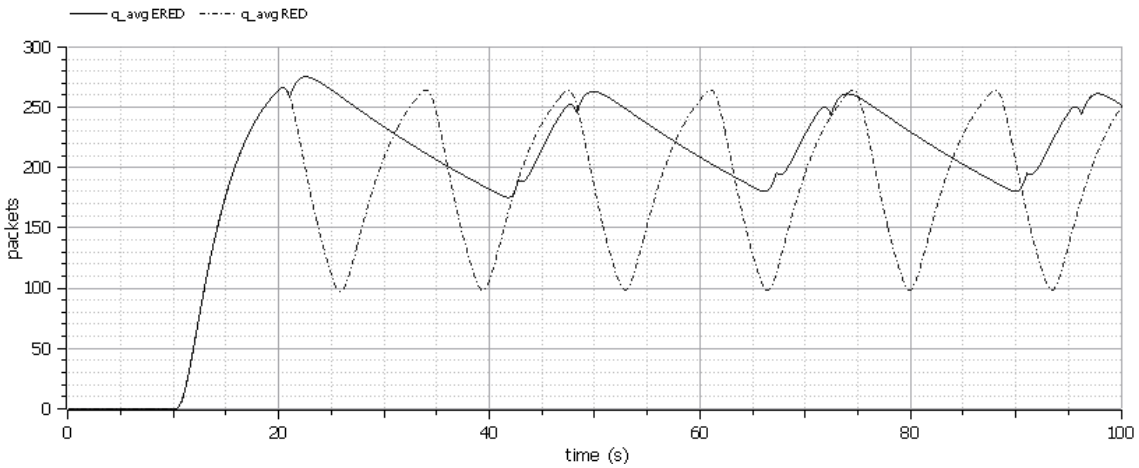


Рисунок 3.12. Различие в поведении $\hat{q}(t)$ в алгоритмах RED и ERED

Рисунок 3.13 демонстрирует поведение системы, работающей по алгоритму DSRED при параметре $\gamma = 0{,}8$. Начальные параметры: $N = 60{,}0; C = 10{,}0; T = 0{,}5; th_{min} = 0{,}25;\ th_{max} = 0{,}5;\ R = 300{,}0;\ w_q = 0{,}0004;\ p_{max} = 0{,}1;\ w_{max} = 32{,}0.$

При сравнении колебаний среднего размера очереди в алгоритмах RED и DSRED (рис. 3.14) видно, что за счет ввода дополнительного порогового сегмента очереди при использовании алгоритма DSRED можно уменьшить амплитуду и частоту колебаний переменной $q(t)$, что демонстрирует явные преимущества данной модификации перед классическим алгоритмом RED.

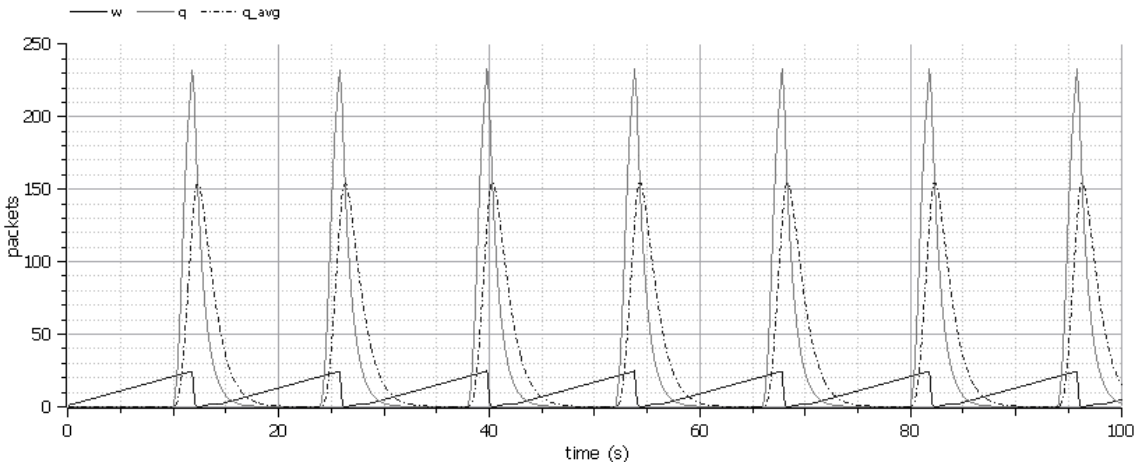


Рисунок 3.13. Динамика изменения $w(t), q(t), \hat{q}(t)$ в системе с управлением типа DSRED

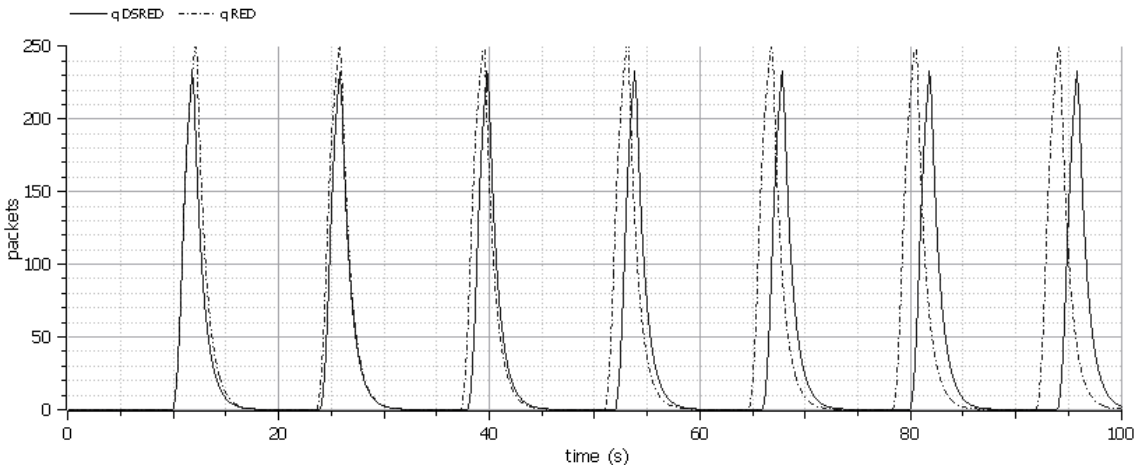


Рисунок 3.14. Различие в поведении функции $q(t)$ в алгоритмах RED и DSRED

Рисунок 3.15 демонстрирует поведение системы, работающей по алгоритму GRED, при следующих начальных параметрах: $N = 60{,}0; C = 10{,}0; T = 0{,}5; th_{min} = 0{,}25;\ th_{max} = 0{,}5;\ R = 300{,}0;\ w_q = 0{,}0004;\ p_{max} = 0{,}1;\ w_{max} = 32{,}0.$

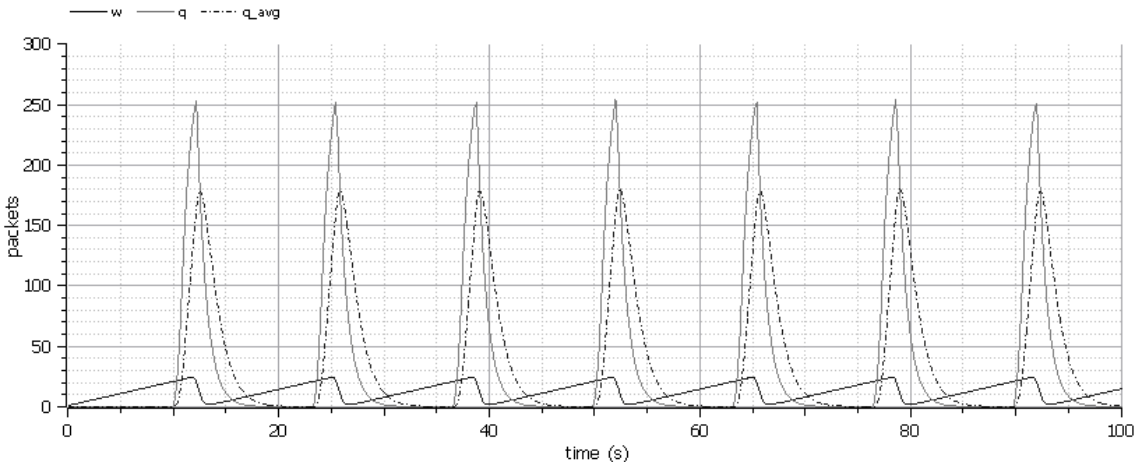


Рисунок 3.15. Динамика изменения $w(t), q(t), \hat{q}(t)$ в системе с управлением по алгоритму GRED

Сравнивая колебания среднего размера очереди в алгоритмах RED и GRED (рис. 3.16 и 3.17), можно сделать вывод о том, что при использовании алгоритма GRED частота колебаний переменной $q(t)$ может быть уменьшена.

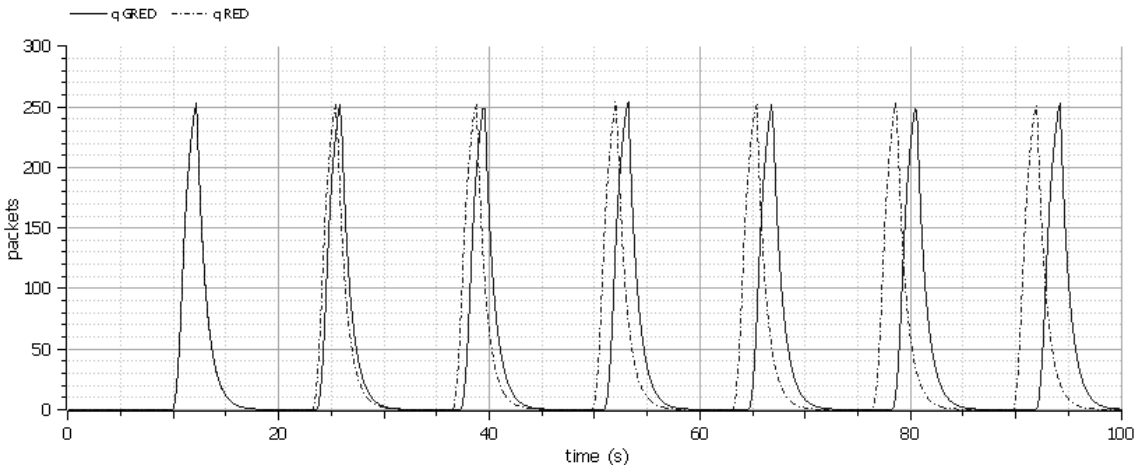


Рисунок 3.16. Различие в поведении среднего размера очереди $q(t)$ в алгоритмах RED и GRED

Изменим параметры системы: $N = 60{,}0; C = 10{,}0; T = 0{,}5; th_{min} = 0{,}5;\ th_{max} = 0{,}9;\ R = 300{,}0;\ w_q = 0{,}0004;\ p_{max} = 0{,}01;\ w_{max} = 32{,}0.$ При увеличении размера очереди, на котором действует функция сброса и при уменьшении доли отбрасываемых пакетов амплитуда колебаний параметров системы увеличилась, а частота заметно уменьшилась, что говорит о том, что данные настройки маршрутизаторов при использовании GRED более предпочтительны, так как параметры системы ведут себя более устойчиво. Результаты представлены ниже (рис. 3.17, 3.18).

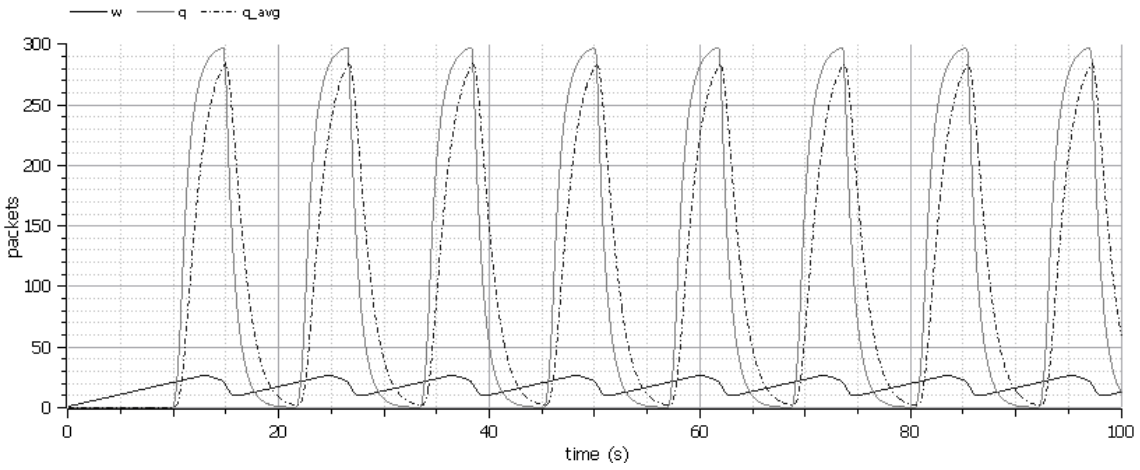


Рисунок 3.17. Динамика изменения $w(t), q(t), \hat{q}(t)$ в системе с управлением типа GRED

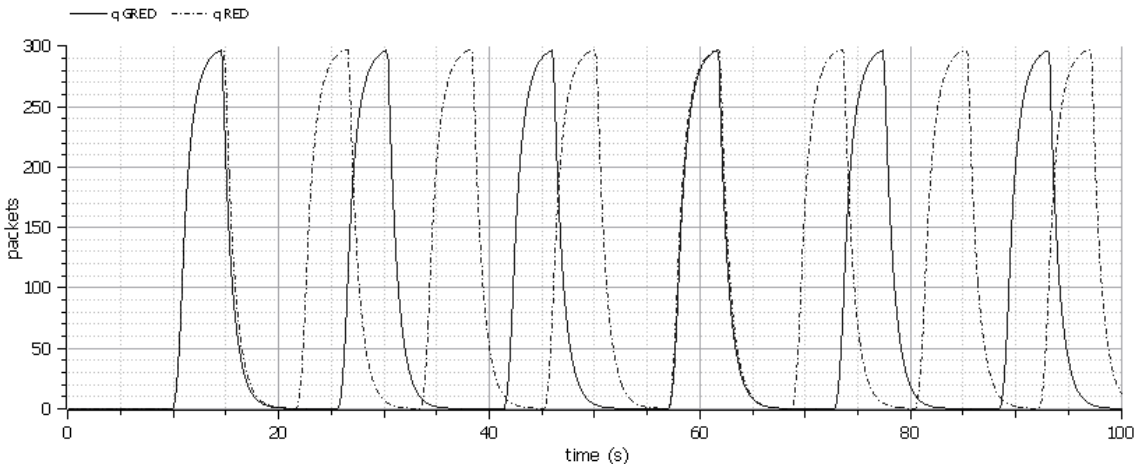


Рисунок 3.18. Различие в поведении среднего размера очереди $q(t)$ в алгоритмах RED и GRED

Полученные результаты говорят о том, что ввод дополнительного сегмента в алгоритме GRED при данных параметрах обеспечивает более плавное поведение функции сброса пакетов за пределами рабочего диапазона, делая такую систему по GRED более устойчивой и плавной по сравнению с системой, работающей по RED.

Рисунок 3.19 демонстрирует поведение системы, работающей по алгоритму SDRED при параметре $N = 0{,}02$, при начальных параметрах: $N = 60{,}0; C = 10{,}0; T = 0{,}5; th_{min} = 0{,}25;\ th_{max} = 0{,}5;\ R = 300{,}0;\ w_q = 0{,}0004;\ p_{max} = 0{,}1;$ $w_{max} = 32{,}0$.

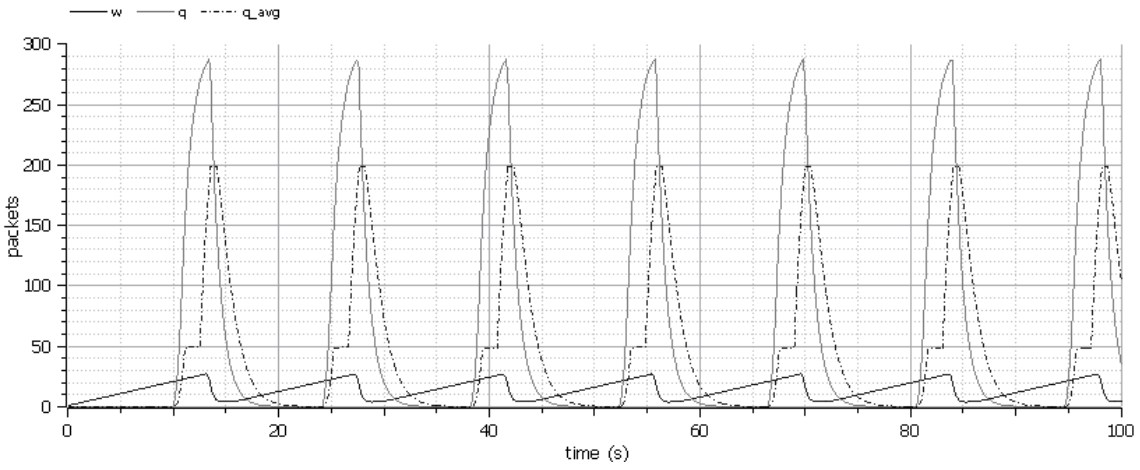


Рисунок 3.19. Динамика изменения $w(t), q(t), \hat{q}(t)$ в системе с управлением по алгоритму SDRED

При сравнении колебаний экспоненциально взвешенной скользящей средней длины очереди в алгоритмах RED и SDRED (рис. 3.20) видно, что при использовании алгоритма SDRED также можно уменьшить частоту колебаний переменной $\hat{q}(t)$.

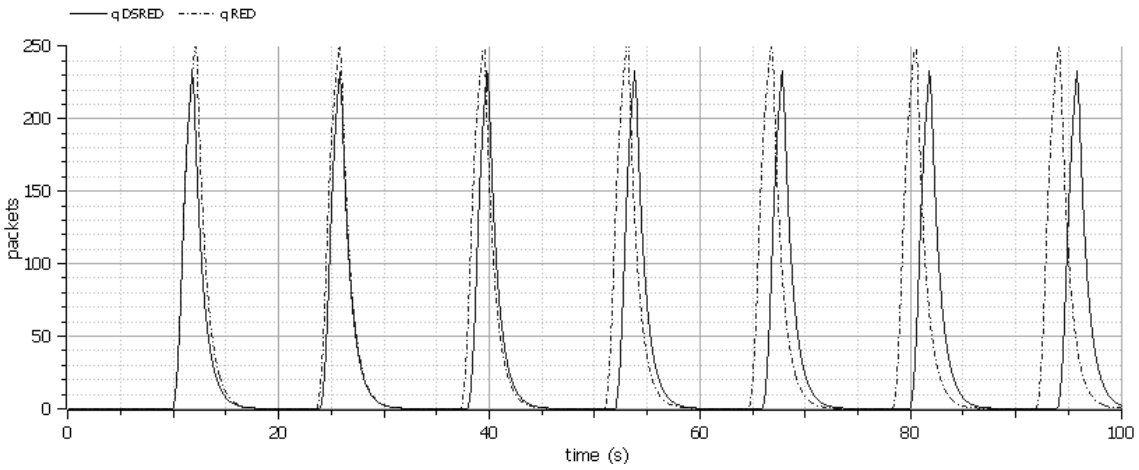


Рисунок 3.20. Различие в поведении параметра $\hat{q}(t)$ в алгоритмах RED и SDRED

### *3.3.2 Результаты имитационного моделирования системы с управлением*

Опишем результаты, полученные в ходе имитационного моделирования гибридной модели протокола TCP Newreno. В качестве изменяемых параметров выступают переменные *number_of_flows* – количество потоков, *RTT* -общее время приема-передачи по всем потокам (в мкс), $Q_{max}$ – максимальный размер очереди (в пакетах).

Результаты моделирования при одном входящем потоке и параметрах *RTT* = 40мкс, максимальном размере очереди 17 пакетов, размер одного пакета 1500 бит представлены на рис. 3.21. Протокол TCP циклически меняет фазы в процессе аддитивного увеличения и мультипликативного уменьшения размера окна. В системе наблюдается устойчивый автоколебательный режим, средний размер TCP окна колеблется от 13 до 23 пакетов, средний размер очереди изменяется о нулевого значения до 16 пакетов.

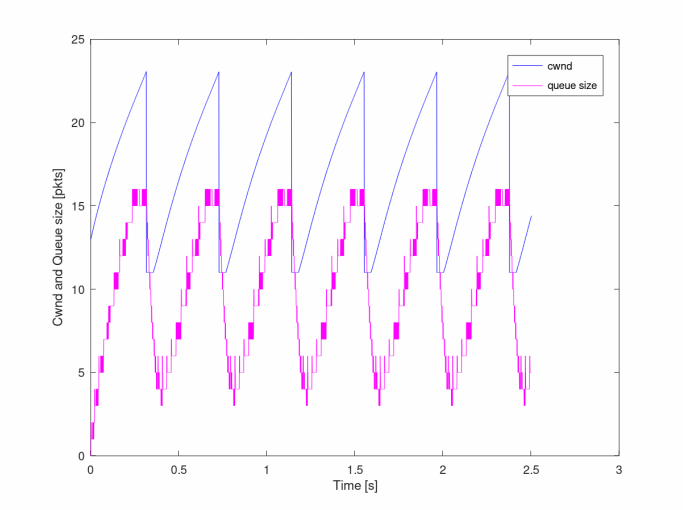


Рисунок 3.21. Поведение функций размера TCP-окна и среднего размера очереди при параметрах $number\_of\_flows = 1, RTT = 40$ *мкс*, $Q_{max} = 17$

При увеличении размера очереди амплитуда колебаний увеличилась, частота колебаний уменьшилась, средний размер TCP окна колеблется от 13 до 31 пакета, средний размер очереди изменяется достигает 29 пакетов (см. рис. 3.22). Аналогично, при еще большем увеличении размера очереди ($Q_{max} = 50$) наблюдаются те же самые тенденции в поведении основных параметров системы (см. рис. 3.23), амплитуда изменений параметров увеличилась, а частота колебаний уменьшилась, что демонстрирует более устойчивый характер поведения системы.

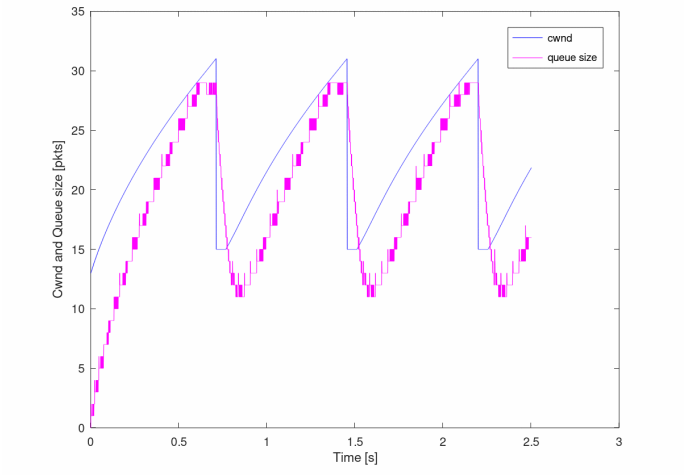


Рисунок 3.22. Поведение функций размера TCP-окна и среднего размера очереди при параметрах $number\_of\_flows = 1, RTT = 40$ *мкс*, $Q_{max} = 30$

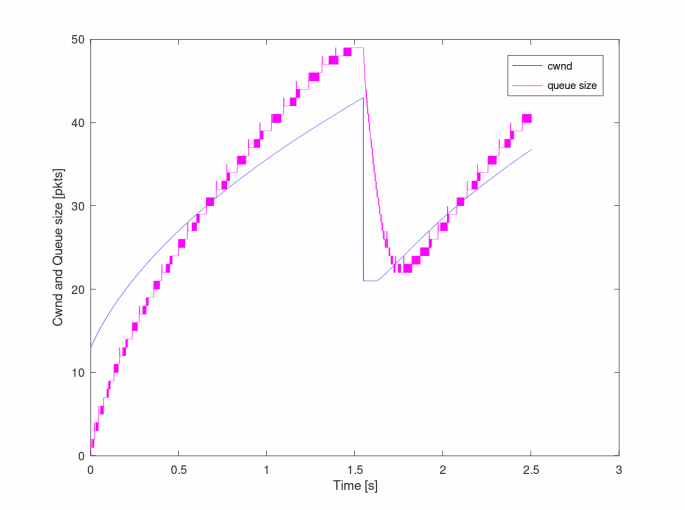


Рисунок 3.23. Поведение функций размера TCP-окна и среднего размера очереди при параметрах $number\_of\_flows = 1, RTT = 40$ *мкс*, $Q_{max} = 50$

Результаты моделирования при двух входящих потоках и параметрах $RTT =$ 40мкс, максимальном размере очереди 50 пакетов, размер пакета 1500 бит представлены на рис. 3.24. В системе также присутствует устойчивый автоколебательный режим, размер TCP-окна колеблется от от 13 до 23 пакетов.

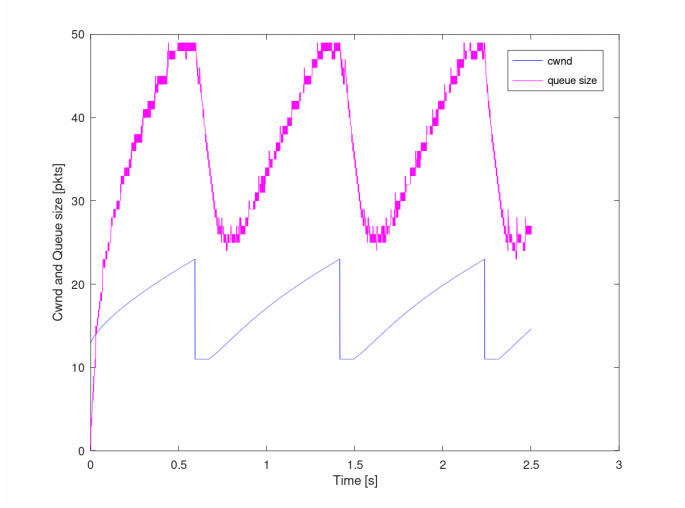


Рисунок 3.24. Поведение функций размера TCP-окна и среднего размера очереди при параметрах $number\_of\_flows = 2, RTT = 40$ *мкс*, $Q_{max} = 50$

При увеличении времени время приема-передачи в два раза ($RTT$ = 80мкс) частота колебаний основных параметров системы уменьшилась, амплитуда TCP-окна изменяется с 13 до 36 (см. рис. 3.25). При уменьшении времени приема-передачи ($RTT$ = 20мкс), частота колебаний увеличилась, амплитуда колебаний уменьшилась: размер очереди колеблется от 28 до 49 пакетов, размер окна меняется

от 10 до 26 пакетов, что говорит о более плавном характере поведения параметров системы (рис. 3.26).

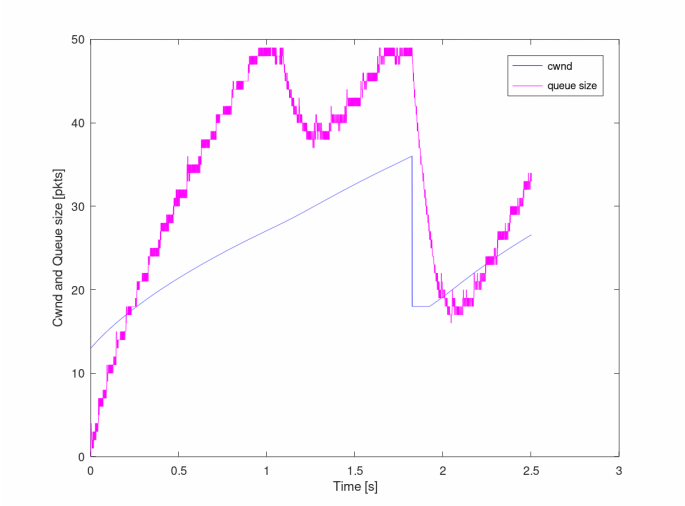


Рисунок 3.25. Поведение функций размера TCP-окна и среднего размера очереди при параметрах $number\_of\_flows = 2, RTT = 80$ *мкс*, $Q_{max} = 50$

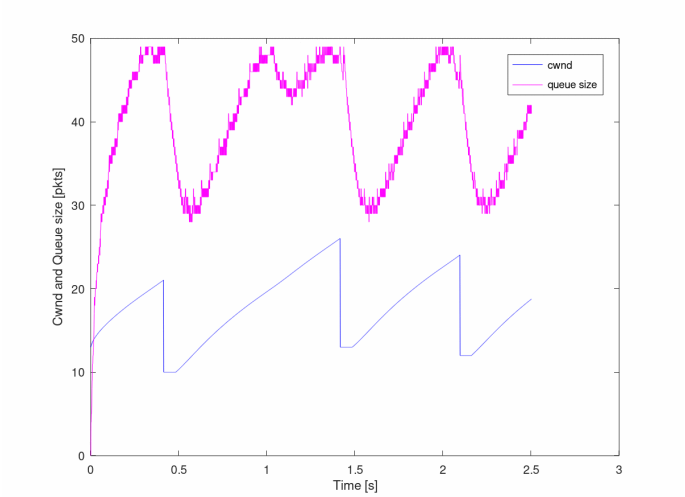


Рисунок 3.26. Поведение функций размера TCP-окна и среднего размера очереди при параметрах $number\_of\_flows = 2, RTT = 20$ *мкс*, $Q_{max} = 50$

Далее представим результаты имитационного моделирования гибридной системы с управлением по алгоритму типа RED в OpenModelica. В качестве изменяемых параметров выступают $Q_{max}$ – максимальный размер очереди, $loss\_factor\_good$ и $loss\_factor\_bad$ – параметры, отражающие нижний и верхний пороговые значения очереди, переменная $T$, отвечающая за физическую задержку передачи пакетов по сети.

Результаты моделирования при параметрах $Q_{max} = 300$, $loss\_factor\_good = 5$, $loss\_factor\_bad = 1$, $T = 0{,}5$ представлены далее. Рисунок 3.27 отражает

поведение среднего размера очереди, а рисунок 3.28 демонстрирует поведение среднего размера TCP, как видно из результатов, система входит в автоколебательный режим, однако алгоритм контролирует как и средней размер очереди, так и средний размер TCP окна, не позволяя этим параметрам достигнуть своего максимального значения и удерживая их на допустимом низком уровне.

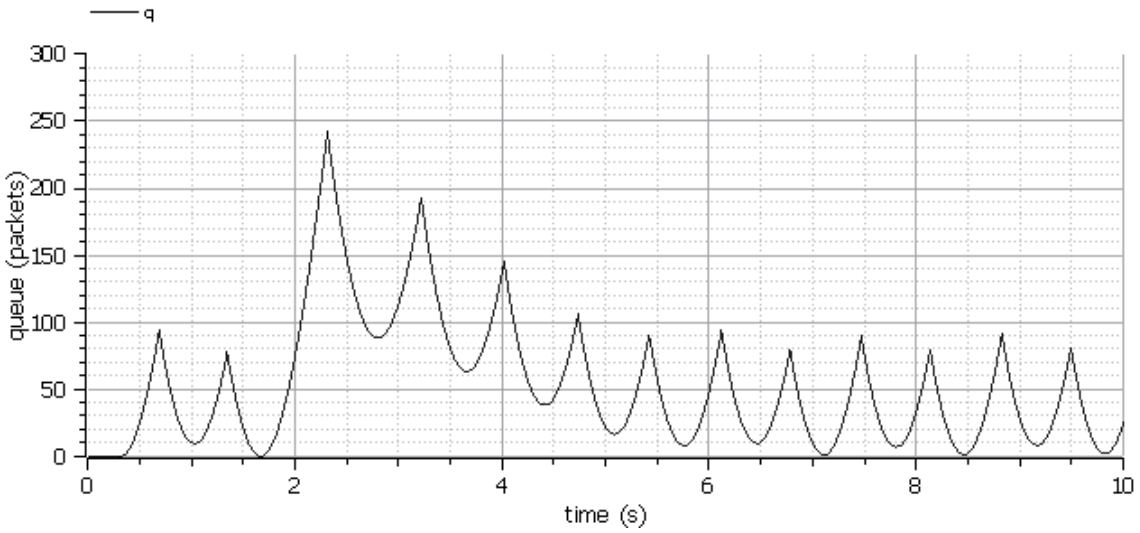


Рисунок 3.27. Средний размер очереди в протоколе TCP Reno при $Q_{max} = 300$, $loss\_factor\_good = 5, loss\_factor\_bad = 1, T = 0{,}5$

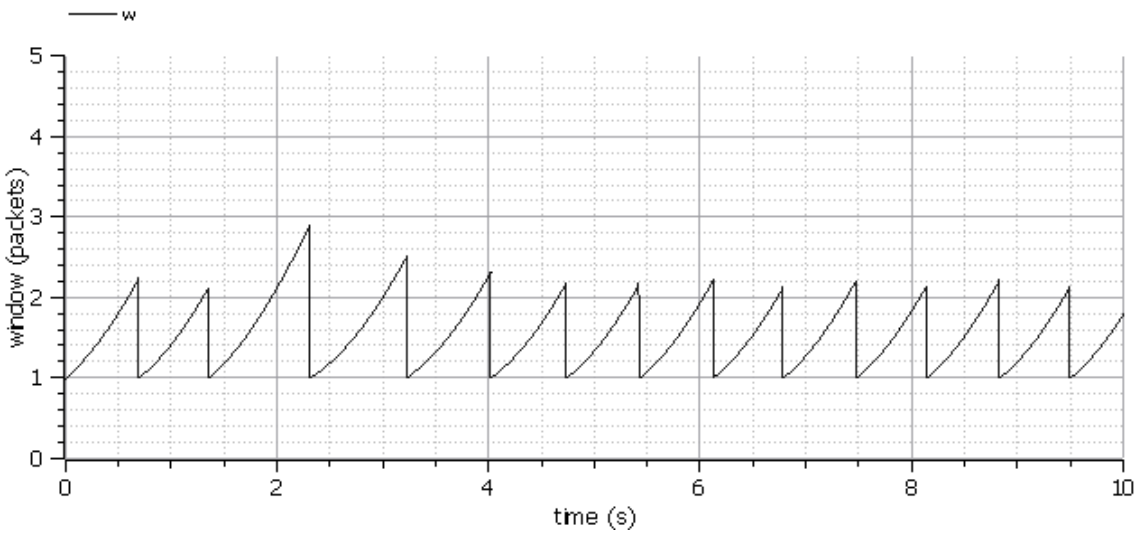


Рисунок 3.28. Средний размер окна в протоколе TCP Reno при $Q_{max} = 300$, $loss\_factor\_good = 5, loss\_factor\_bad = 1, T = 0{,}5$

Уменьшим промежуток, на котором действует функция сброса пакетов. Результаты моделирования при параметрах $Q_{max} = 300$, $loss\_factor\_good = 2$, $loss\_factor\_bad = 1, T = 0{,}5$ представлены ниже. Амплитуда колебаний среднего размера очереди (рис. 3.29) уменьшилась, что говорит о более устойчивом

поведении параметров системы, однако частота колебаний среднего размера окна (рис. 3.30) незначительно увеличилась.

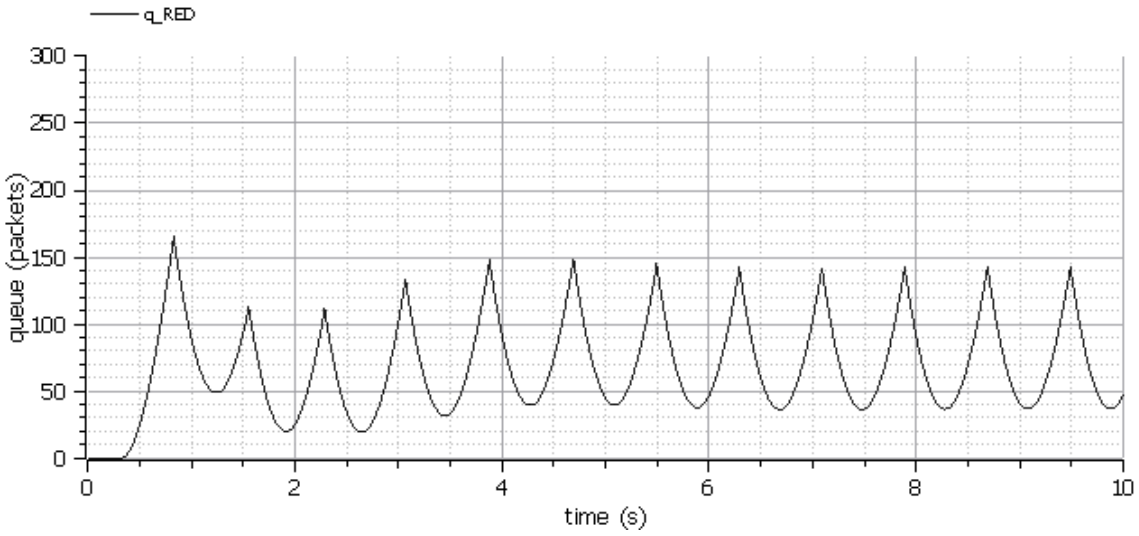


Рисунок 3.29. Средний размер очереди в протоколе TCP Reno при $Q_{max} = 300$, $loss\_factor\_good = 2, loss\_factor\_bad = 1, T = 0{,}5$

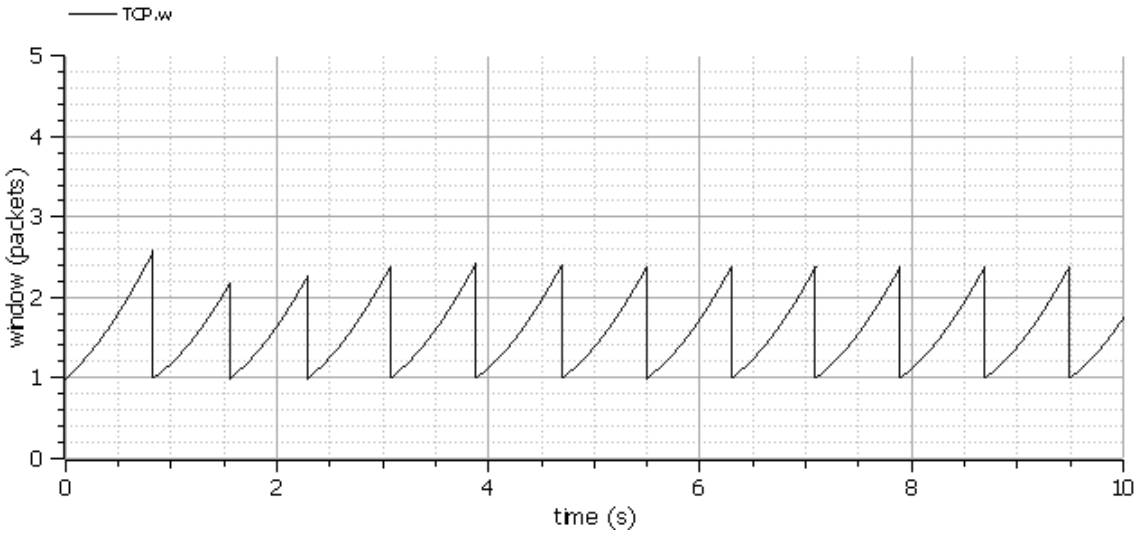


Рисунок 3.30. Средний размер окна в протоколе TCP Reno при $Q_{max} = 300$, $loss\_factor\_good = 2, loss\_factor\_bad = 1, T = 0{,}5$

Продемонстрируем, как изменяются параметры системы при увеличении времени задержки $T$. Графики (рис. 3.31, 3.32) отражают поведение параметров системы при параметрах $Q_{max} = 300,\ loss\_factor\_good = 5, loss\_factor\_bad = 1, T = 0{,}6$. Амплитуда и частота функции среднего размера очереди уменьшилась, что говорит о более стабильном поведении данной функции при вышеописанных параметрах.

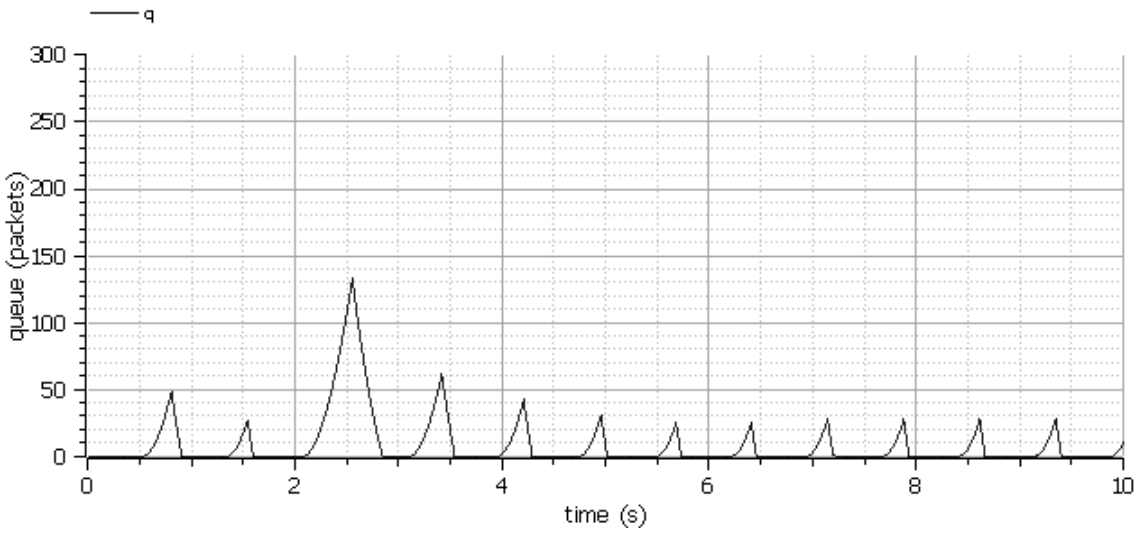


Рисунок 3.31. Средний размер очереди в протоколе TCP Reno при $Q_{max} = 300$, $loss\_factor\_good = 5, loss\_factor\_bad = 1, T = 0{,}6$

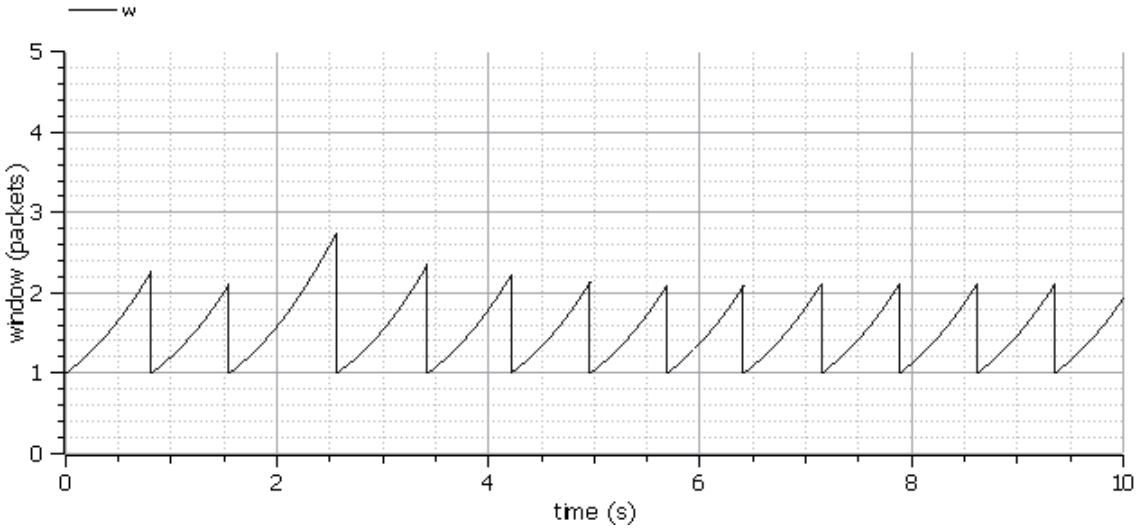


Рисунок 3.32. Средний размер окна в протоколе TCP Reno при $Q_{max} = 300$, $loss\_factor\_good = 5, loss\_factor\_bad = 1, T = 0{,}6$

## Заключение

В ходе данной работы был описан принцип действия системы с алгоритмом управления трафиком типа RED, представлена математическая модель данной системы. Так же были описаны такие модификации алгоритма RED, как ERED, DSRED, GRED и SDRED. В среде OpenModelica на языке объектно-ориентированного программирования Modelica проведено численное моделирование систем с управлением, в качестве которых выступали вышеописанные системы. В результате данного моделирования были получены графики изменений основных параметров систем, отражающие то, что данная система с управлением при некоторых начальных параметрах переходит в автоколебательный режим функционирования. Результаты демонстрируют различие в поведении параметров системы при определенных начальных параметрах, а также некоторые преимущества, которые могут быть получены при использовании определенных модификаций алгоритма RED. Системы, функционирующие по ERED, GRED и SDRED, в большинстве случаев, демонстрировали уменьшение частоты колебаний, а алгоритм DSRED в большинстве проводимых экспериментов уменьшал не только частоту колебаний, но и амплитуду, делая характер поведения системы более стабильным. Также большинство систем положительно реагировали на увеличении длины очереди, на котором работает функция сброса пакетов, демонстрируя уменьшение частоты и амплитуды колебаний параметров системы, уменьшение доли отбрасываемых пакетов аналогично, в большинстве случаев, делало режим функционирования систем более плавным и стабильным.

При подготовке имитационной модели была изучена и усовершенствована существующая программа модели на языке Matlab. В среде Octave было проведено имитационное моделирование гибридной модели системы, работающей по протоколу TCP Newreno. Гибридная модель описывала переход TCP состояний системы и обработку очередей в сети. Полученные результаты имитационного моделирования так же, как и результаты численного моделирования, демонстрируют наличие в системе устойчивого автоколебательного режима, параметры колебаний которого зависят от начальных параметров.

Для верификации полученных в ходе численного моделирования результатов, основываясь на модель в среде Octave, в среде OpenModelica была разработана

имитационная модель системы, функционирующая по протоколу TCP Reno и обрабатывающая очереди в соответствии с алгоритмом типа RED. Подтверждая результаты математической модели, в большинстве проводимых экспериментов в системе наблюдались автоколебания, частота и амплитуда которых уменьшалась при увеличении промежутка длины очереди, на котором работает функция сброса пакетов, демонстрируя, что при некоторых начальных значениях параметров функционирование системы можно сделать более устойчивым и плавным.

В ходе дальнейшего изучения данной темы планируется нахождение и исследование областей возникновения автоколебаний системы с управлением по алгоритму RED и его модификациям.

## Список литературы


1. Апреутесей А.М.Ю. Простая модель системы с активным управлением очередью по алгоритму типа RED // Информационно-телекоммуникационные технологии и математическое моделирование высокотехнологичных систем: материалы Всероссийской конференции с международным участием. Москва, РУДН, 16–20 апреля 2018 г. – Москва: РУДН, 2018. – с. 285-287.
2. Апреутесей, А.М.Ю., Королькова, А.В, Кулябов, Д.С. Моделирование системы с управлением на языке Modelica // Системы управления, технические системы: устойчивость, стабилизация, пути и методы исследования. Материалы молодежной секции в рамках IV Международной научно-практической конференции. Елец, ЕГУ им. И. А. Бунина, 25 апреля 2018 г. – Елец: ЕГУ им. И. А. Бунина, 2018. – с. 108-112.
3. Apreutesey A.M.Y., Korolkova A.V. A Simple Model of Active Queue Management System According to the RED Algorithm // 7th International conference “Problems of Mathematical Physics and Mathematical Modelling”: Books of abstracts. Moscow, NRNU MEPhI, 25–27 June 2018 - Moscow, NRNU MEPhI, 2018 – p. 155-157
4. Апреутесей А.Ю., Завозина А.В., Королькова А.В., Кулябов Д.С. Вычислительная и имитационная модели системы с управлением на Modelica // Вестник РУДН. Серия «Математика. Информатика. Физика». - 2018. - № 4. – с. 357-370.
5. Апреутесей А. М. Ю., Королькова А.В., Кулябов Д.С. Моделирование модификаций алгоритма RED в среде OpenModelica // Информационно-телекоммуникационные технологии и математическое моделирование высокотехнологичных систем: материалы Всероссийской конференции с международным участием. Москва, РУДН, 15–19 апреля 2019 г.. – Москва: РУДН, 2019. – с. 398-406.
6. Floyd S., Jacobson V. Random Early Detection gateways for congestion avoidance// IEEE/ACM Transactions on Networking, – 1993. – Vol. 1, №4. p. 397-413.
7. Misra V., Gong W.-B., Towsley D. Stochastic Differential Equation Modeling and Analysis of TCP-Window size Behavior. – 1999. – Vol.99.

8. Misra V., Gong W.-B., Towsley D. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED // ACM SIGCOMM Computer Communication Review 30 (4), – p. 151-160.

9. Hollot C.V., Misra V., Towsley D., Gong W. B. Analysis and Design of Controllers for AQM Routers Supporting TCP flows. – 2002

10. Liu Y., Presti F.L., Misra V., Towsley D., Gu Y. Fluid models and solutions for large-scale IP networks // Proceeding SIGMETRICS'03 Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems. – p. 91-101

11. Kunniyur S., Srikant R. Analysis and design of an adaptive virtual queue algorithm for active queue management. In Proceedings of ACM/SIGCOMM'2001, 2001.

12. van Foreest N. D. Queues with congestion-dependent feedback: Ph.D. thesis. — Enschede: University of Twente, 2004.

13. Охотников С.С. Обеспечение устойчивости TCP сессий маршрутизатором RED // Научный журнал «Информатика и системы управления». – 2007. - №2.

14. Korolkova A. V., Velieva T. R., Abaev P. O., Sevastianov L. A., Kulyabov D. S. Hybrid simulation of active traffic management // Proceedings 30th European Conference on Modelling and Simulation. – 2016. p. 685-691

15. Королькова А.В. Методы моделирования сложных систем // Материалы Международной научно-практической конференции «Системы управления, технические системы: устойчивость, стабилизация, пути и методы исследования» (6 апреля 2016 г.). Елец: Елецкий государственный университет им. И. А. Бунина. с.79–96.

16. Королькова А.В., Велиева Т.Р. Гибридный подход к моделированию сложных систем // Материалы Международной научно-практической конференции «Системы управления, технические системы: устойчивость, стабилизация, пути и методы исследования материалы научно-практического семинара молодых ученых и студентов. 2017. с. 19-27

17. Колесов Ю.Б., Сениченков Ю.Б., Уркия А., Мартин-Виллалба К. Гибридные системы. Сравнительный анализ языков моделирования Modelica и Model Vision Language // Университетский научный журнал. 2014. № 8. с. 102—111.

18. Breitenecker F. Classification and evaluation of features in advanced simulators // Proc. MATHMOD-09. –Vienna, 2009.

19. Jacobson V. Congestion Avoidance and Control // SIGCOMM ’88: Symposium Proceedings on Communications Architectures and Protocols. — New York, NY, USA: ACM, 1988. — p. 314–329.

20. Allman M., Paxson V., Stevens W. TCP Congestion Control. RFC-2581 — United States: RFC Editor, 1999.

21. Braden R. Requirements for Internet Hosts — Communication Layers. RFC-1122 — United States: RFC Editor, 1989.

22. Floyd S. Explicit Congestion Notification (ECN) Mechanism in the TCP/IP Protocol // ACM Computer Communications Review. — 1994. — Vol. 24.

23. Ramakrishnan K., Floyd S., Black D. The Addition of Explicit Congestion Notification (ECN) to IP. RFC-3168 — United States: RFC Editor, 2001

24. Башарин Г.П., Гайдамака Ю.В., Самуйлов К.Е., Яркина Н.В.. Модели для анализа качества обслуживания в сетях связи следующего поколения //Уч. пособие. Москва, ИПК РУДН, 2008, с. 85-86.

25. Королькова А. В., Кулябов Д. С. Математическая модель динамики поведения параметров систем типа RED // Вестник РУДН. Серия «Математика. Информатика. Физика».– 2010. – № 2(1). – с. 54-64.

26. Королькова А. В., Кулябов Д. С. Предварительная классификация алгоритмов семейства RED // Материалы XII научной конференции МГТУ «Станкин» и «Учебно-научного центра математического моделирования МГТУ «Станкин» – ИММ РАН» по математическому моделированию и информатике: Программа. Сборник докладов. / под ред. О. Казакова. — М.: ИЦ ГОУ ВПО МГТУ «Станкин», 2009. — с. 125–128.

27. Королькова А.В., Кулябов Д.С., Черноиванов А.И. К вопросу о классификации алгоритмов RED // Вестник РУДН, серия «Математика. Информатика. Физика». — 2009. — № 3. — с. 34-46.

28. Que D., Chen Z., Chen B. An Improvement Algorithm Based on RED and Its Performance Analysis // IEEE, 2008. — p. 2005–2008.

29. Zheng B., Atiquzzaman M. DSRED: A New Queue Management Scheme for the Next Generation Internet // IEICE TRANS. COMMUN. — 2006. — Vol. E89–B, No 3.

30. Iannaccone G., May M., Diot C. Aggregate traffic performance with active queue management and drop from tail // SIGCOMM Comput. Commun. Rev. — 2001. — Vol. 31, № 3. — p. 4–13. — ISSN 0146-4833.

31. Ryoo I., Yang M. A State Dependent RED: An Enhanced Active Queue Management Scheme for Real-Time Internet Services // IEICE Trans. Commun. — 2006. — Vol. E89–B, No 2. — p. 614–617.

32. Modelica Language Specification, Version 3.3. Modelica Association (May 9, 2012) // Режим доступа: URL: https://www.modelica.org/documents/ ModelicaSpec33.pdf. — (дата обращения: 20.05.2018).

33. Kilduff R. J. Analysis of Congestion Models for TCP Networks — November, 2003

34. Bohacek S., Hespanha J., Lee J., Obraczka K. A Hybrid Systems Modeling Framework for Fast and Accurate Simulation of Data Communication Networks // Proceedings of ACM Sigmetrics, 2003.

35. Fall K., Floyd S., Simulation-Based Comparisons of Tahoe, Reno and SACK TCP, ACM SIGCOMM Computer Communication Review 26 (3) (1996) 5‑21. doi:10.1145/235160.235162.

36. Королькова А. В. Определение области возникновения автоколебаний в системах типа RED // Вестник РУДН. Серия «Математика. Информатика. Физика». — 2010. — № 1. — С. 103–105.

## Приложение 1

### Программа для ЭВМ «Численное моделирование системы с управлением по алгоритму RED»

В разработке программы участвовали А.М.Ю. Апреутесей, А.В. Королькова.

Полный листинг программы:

```
function qAdd
  input Real q;
  input Real w;
  input Real T;
  input Real C;
  input Real N;
  input Real R;
  output Real qOut;
  protected Real q1;
  protected Real q2;
algorithm
  q1 := N * w / T - C;
  q2 := q + q1;
  qOut := if q2 > R then R - q else if q2 > 0.0 then q1 else
-q;
end qAdd;

function wAdd
  input Real wIn;
  input Real wmax;
  input Real T;
  output Real wOut;
algorithm
  wOut := if noEvent(wIn >= wmax) then 0.0 else 1.0 / T;
end wAdd;

class Red
  parameter Real N(start = 60.0) "Количество сессий";
  parameter Real c(start = 10.0) "Интенсивность
обслуживания, Mbps";
  parameter Real packet_size(start = 500.0) "Размер пакета,
bit";
  parameter Real T(start = 0.05) "Время двойного оборота";
  parameter Real thmin(start = 0.25) "Нормализованный нижний
порог";
  parameter Real thmax(start = 0.5) "Нормализованный верхний
порог";
  parameter Real R(start = 300.0) "Размер очереди";
  parameter Real wq(start = 0.0004) "Параметр EWMS";
  parameter Real pmax(start = 0.1) "Максимальная вероятность
сброса";
  parameter Real wmax(start = 32.0) "Максимальный размер
окна";
```

```
      Real p(start = 0.0) "Вероятность сброса";
      Real w(min = 1.0, max = wmax, start = 1.0, fixed = true)
"Окно";
      Real q(max = R, start = 0.0, fixed = true) "Мгновенная
длина очереди";
      Real q_avg(start = 0.0) "EWMS длины очереди";
      Real C = 125000.0 * c / packet_size "Интенсивность
обслуживания, packets";
    equation
      der(w) = wAdd(w, wmax, T) + (-0.5) * w * delay(w, T, T) *
delay(p, T, T) / delay(T, T, T);
      der(q) = qAdd(pre(q), w, T, C, N, R);
      der(q_avg) = wq * C * (q - q_avg);
      p = if q_avg < thmin * R then 0.0 else if q_avg > thmax *
R then 1.0 else (q_avg / R - thmin) * pmax / (thmax - thmin);
      when w <= 1.0 then
        reinit(w, 1.0);
      end when;
      when q >= R then
        reinit(q, R);
      end when;
    end Red;
```

## Приложение 2

### Программа для ЭВМ «Имитационное моделирование гибридной модели протокола TCP»

В разработке программы участвовали А.М.Ю. Апреутесей, А.В. Королькова.

Полный листинг программы:

```
function x_TCP = TCP_hyb_simulation()
clear all;
global Bndwdth;
Bndwdth = 10000000/(1500*8); % 10Mb, 12000 packets
Window_growth = 1;
global flows_num;
flows_num = 2;

global RTTIME_FLOW;
RTTIME_FLOW = [];
RTTIME_FLOW(1,1) = 0.010;
RTTIME_FLOW(2,1) = 0.010;
global RTTIME;
RTTIME = [];
RTTIME(1:flows_num,1) = RTTIME_FLOW(1:flows_num,1);
DDD = RTTIME_FLOW(1);

Tf = 2.5;  T0 = 0;

global ST;
ST=0.001; %1/(10*Bndwdth);
time = [0];
timer0 = zeros(flows_num,1);
timer = [];
timer(:,1)=timer0;

tdot = [];
tdot = timer0;
w0 = 13*ones(flows_num,1);

global w;
w = [];
w(:,1)=w0;
r_wind0 = 0*[1:flows_num]';

global r_wind;
r_wind = [];
r_wind(:,1) = r_wind0;

global amount;
amount = 1;
filter_amount = 0;
```

```
      global b;
      b = 1.45;
      L=1;
      Wth = [];
      Wth0 = 2000*[1:flows_num]';
      Wth(:,1) = Wth0;
      m = 2;
      ndrops = 1;

      global state;
      state = [];
      for i = 1:flows_num state{i} = 'C_Avoidance'; end
      global queue_state;
      queue_state = 'QUEUE_N_FULL';
      global Qmax;
      Qmax = 50 - 1 %Queue size in ns is number of packets queue
can hold + 1
      q0 = 0*[1:(Qmax)]';
      global q; %queue
      q = [];
      q(:,1)=q0;
      global qptr;
      qptr = 0;
      % states [r1,..rn]
      rq0 = 0*[1:(flows_num)]';
      global rq;
      rq = [];
      rq(:,1)=rq0;
      global drops;
      drops = [];
      drops0 = 0*[1:(flows_num)]';
      drops(:,1) = drops0;
      global dropstime;
      pack_drop_time = [];

      global tq;
      tq = [];
      tq(1) = 1/Bndwdth;

      global te;
      te = [];
      te(1) = 0;

      global enamount;
      enamount = 1;
      global enpacket;
      global k;
      k = [];
      k0 = 0*[1:(flows_num)]';
      k(:,1) = k0;
      enpacket = 0; %???
      global wtmp;
      wtmp = [];
```

```
      wtmp0 = 0*[1:(flows_num)]';
      wtmp = wtmp0;
      global timertmp;
      global border;
      border = [];
      b0 = 0*[1:(flows_num)]';
      border(:,1) = b0;
      global lastw;
      lastw = [];
      l0 = 1*[1:(flows_num)]';
      l_wind(:,1) = l0;
      timeamount =1;

      for i = T0:ST:Tf;
        if amount*ST > amount_time
        amount*ST
        amount_time = amount_time + 1;
      end
      for j = 1:flows_num
      switch state{j}
      case 'S_START'
        timer(j,amount+1) = 0;
        wdot = (log(m)/RTTIME(j,amount))*w(j,amount);
        w(j,amount+1) = w(j,amount)+wdot*ST; %Euler
        r_wind(j,amount+1) = (b*w(j,amount)/RTTIME(j,amount));
        % Check for Congestion Avoidance
        if w(j,amount+1) >= Wth(j);
              w(j,amount+1) = Wth(j);
              state{j} = 'C_Avoidance';
        end
        % Check for Slow Start Delay
        if drops(j) > 0
        %w(flows_num+1,amount) = DDD;
              timer(j,amount+1) = RTTIME(j,amount);
              wtmp(j) = w(j,amount+1);
              state{j} = 'S_START_DELAY';
        end;
        queue(j);
      case 'S_START_DELAY'
        tdot = -1;
        wdot = (log(m)/RTTIME(j,amount))*w(j,amount);
        w(j,amount+1) = w(j,amount)+wdot*ST;
        timer(j,amount+1) = timer(j,amount)+tdot*ST;
        r_wind(j,amount+1) = (b*(w(j,amount+1) -drops(j) -
drops(j)/(wtmp(j)+1))/RTTIME(j,amount));
        if timer(j,amount+1) < 0;
              state{j} = 'F_RECOVERY';
              if drops(j) > w(j,amount+1)/2 + 1
              k(j) = 1+ceil(log2(drops(j)));
              else
              k(j) =
ceil(log2((1+(w(j,amount)/2))/(1+(w(j,amount)/2)-drops(j))));
              end
```

```
            r_wind(j,amount+1) = (1+(w(1,amount)/2)-
drops(j))/RTTIME(j,amount);
            w(j,amount+1) = floor(w(j,amount)/2);
            timer(j,amount+1) = RTTIME(j,amount);
            timertmp=RTTIME(j,amount)*k;
            drops(j) = 0;
      elseif w(j,amount+1) < max(2+drops(j), 2*drops(j)-4)
            timer(j,amount+1) =1;
            w(j,amount+1) = floor(w(j,amount+1)/2);
            drops(j) = 0;
            state{j} = 'T_OUT';
      end
      queue(j);
     case 'F_RECOVERY'
      tdot = -1;
      wdot = 0;
      r_winddot = 0;
      w(j,amount+1)=w(j,amount)+wdot*ST;
      r_wind(j,amount+1) = r_wind(j,amount)+r_winddot*ST;
      timer(j,amount+1) = timer(j,amount)+tdot*ST;
      % Check Congestion Avoidance
      if timer(j,amount+1) < 0 & k(j) > 0
            timer(j,amount+1) = RTTIME(j,amount);
            k(j) = k(j) - 1;
            if k(j) <= 0
                  drops(j) = 0;
                  state{j} = 'C_Avoidance';
            end
            r_wind(j,amount+1) = 2*r_wind(j,amount+1);
      end
      queue(j);
     case 'T_OUT'
      tdot = -1;
      wdot = 0;
      r_winddot = 0;
      w(j,amount+1)=w(j,amount)+wdot*ST;
      r_wind(j,amount+1)=0;
      timer(j,amount+1) = timer(j,amount)+tdot*ST;
      % Check for Slow Start
      if timer(j,amount+1) < 0
            state{j} = 'S_START';
            Wth(j) = w(j,amount+1)/2;
            w(j,amount+1)=1;
      end;
      queue(j);
     case 'C_Avoidance'
      timer(j,amount+1) = 0;
      wdot = (L/(RTTIME(j,amount)));
      w(j,amount+1)=w(j,amount)+wdot*ST;
      r_wind(j,amount+1)=w(j,amount)/RTTIME(j,amount);
      if drops(j) > 0
            timer(j,amount+1) = RTTIME(j,amount);
            wtmp(j) = w(j,amount+1);
```

```
            state{j} = 'C_AVOIDANCE_DELAY';
       end
       queue(j);
     case 'C_AVOIDANCE_DELAY'
       tdot = -1;
       wdot = (L/RTTIME(j,amount));
       w(j,amount+1)=w(j,amount)+wdot*ST;
       r_wind(j,amount+1)=(w(j,amount) -1 -
drops(j)/(wtmp(j)+1))/RTTIME(j,amount);
       timer(j,amount+1)=timer(j,amount)+tdot*ST;
       if timer(j,amount+1) < 0;
            state{j} = 'F_RECOVERY';
            if drops(j) > w(j,amount+1)/2 + 1
                  k(j) = 1+ceil(log2(drops(j)));
            else
                  k(j) =
ceil(log2((1+(w(j,amount)/2))/(1+(w(j,amount)/2)-drops(j))));
            end
            r_wind(j,amount+1) = (1+(w(j,amount)/2)-
drops(j))/RTTIME(j,amount);
            w(j,amount+1) = floor(w(j,amount)/2);
            timer(j,amount+1) = RTTIME(j,amount);
            drops(j) = 0;
            l_wind(j,1) = 1;
            %check for TimeOut
       elseif w(j,amount+1) < max (2+drops(j), 2*drops(j)-4)
            timer(j,amount+1) = 1;
            w(j,amount+1) = floor(w(j,amount)/2);
            drops(j) = 0;
            state{j} = 'T_OUT';
       end
       queue(j);
     end;
     end;
     amount = amount + 1;
     end;
     output;

     function queue(j)
     global flows_num;
     global Bndwdth;
     global amount;
     global q;
     global rq;
     global tq;
     global r_wind;
     global drops;
     global ST;
     global pack_drop_time;
     global RTTIME;
     global RTTIME_FLOW;
     global qptr;
     global Qmax;
```

```
      global state;
      global queue_state;
      global w;
      global te;
      global enamount;
      global enpacket;
      persistent rqdot;
      global l_wind;
      global border;
      global wtmp;
      global b;
      if strcmp(state(j),'C_AVOIDANCE_DELAY')
        rqdot = (floor(w(j,amount+1)) -drops(j) -1-
drops(j)/(floor(wtmp(j))+1))/RTTIME(j,amount);
      else
        rqdot = (floor(w(j,amount+1)) -1)/RTTIME(j,amount);
      end
      % Integrate fluid F_Recoveryom flows
      rq(j,amount+1) = rq(j,amount) + rqdot*ST;
      if (j==1)
        q(:,amount+1) = q(:,amount);
        te(amount+1) = te(amount);
        tq(amount+1) = tq(amount);
        % Service the Queue
      if qptr > 0
        tqdot = -1;
        tq(amount+1) = tq(amount) + tqdot*ST;
        if tq(amount+1) <= 0
              q(1:min(qptr,Qmax-1), amount+1) =
q(2:min(qptr+1,Qmax),amount+1);
              q(min(qptr+1,Qmax):Qmax, amount+1) = 0;
              qptr = qptr -1;
              tq(amount+1) = 1/Bndwdth;
        end
        if qptr < Qmax
        queue_state = 'QUEUE_N_FULL';
        end
      end
      end
      if w(j,amount+1) >= l_wind(j)
        border (j)= 1;
        l_wind(j) = floor(w(j,amount+1))+1;
      end
      if qptr < Qmax && border(j) == 1
        if qptr < Qmax - 1
              qptr = qptr + 2;
              q(qptr-1:qptr,amount+1) = j;
              rq(j,amount+1) = rq(j,amount+1) - 1;
        else % qptr = Qmax - 1
              qptr = qptr + 1;
              q(qptr,amount+1) = j;
              drops(j) = drops(j) + 1;
              pack_drop_time = [pack_drop_time,[amount*ST,j]'];
```

```
        rq(j,amount+1) = rq(j,amount+1) -1;
 end
 border(j) = 0;
elseif qptr == Qmax && border(j) == 1
 drops(j) = drops(j) + 2;
 pack_drop_time = [pack_drop_time,[amount*ST,j]'];
 rq(j,amount+1) = rq(j,amount+1) - 1;
 border(j) = 0;
end
if rq(j,amount) >= 1
 if qptr < Qmax
        qptr = qptr + 1;
        q(qptr,amount+1) = j;
 else
        drops(j) = drops(j) + floor(rq(j,amount+1));
        pack_drop_time = [pack_drop_time,[amount*ST,j]'];
 end
 rq(j,amount+1) = rq(j,amount+1) - 1;
end
RTTIME(j,amount+1) = RTTIME_FLOW(j) + qptr/Bndwdth;
% End of function Queue

function output()
global amount;
global ST;
global w;
global r_wind;
global q;
global rq;
global RTTIME;
global RTTIME_FLOW;
global pack_drop_time;
global flows_num;
global te;
hold off
plot([1:amount]*ST,(w(1,:)),'b')
xlabel('Time [s]');
ylabel('Cwnd and Queue size [pkts]');

hold on
%plot([1:amount]*ST,(w(2,:)),'c')
plot([1:amount]*ST,sum(q(:,1:amount)~=0),'m')
legend( 'cwnd', 'queue size');

%plot([1:amount]*ST,te(1:amount)*1000,'k');
%plot([1:amount]*ST,RTTIME(1,1:amount)*1000,'g');
if length(pack_drop_time) > 0
 for i = 1:length(pack_drop_time)
        switch round(pack_drop_time(2,i))
        case 1
              %plot(pack_drop_time(1,i),14,'b+');
        case 2
              %plot(pack_drop_time(1,i),14,'c+');
```

```
        end
  end
end
hold off
```

## Приложение 3

### Программа для ЭВМ «Имитационное моделирование гибридной системы с управлением по алгоритму типа RED в OpenModelica»

В разработке программы участвовали А.М.Ю. Апреутесей, А.В. Королькова.

Полный листинг программы:

```
class RED_simulation
  Real router1.i.rate;
  Boolean router1.i.drop;
  Real router1.i.RTT(start = 0.02);
  Real router1.o.rate;
  Boolean router1.o.drop;
  Real router1.o.RTT(start = 0.02);
  Real router2.i.rate;
  Boolean router2.i.drop;
  Real router2.i.RTT(start = 0.02);
  Real router2.o.rate ;
  Boolean router2.o.drop;
  Real router2.o.RTT(start = 0.02);
  Real TCP.o.rate;
  Boolean TCP.o.drop;
  Real TCP.o.RTT(start = 0.02);
  constant Integer TCP.TCPState_s_start_mode = 1;
  constant Integer TCP.TCPState_fast_recovery_mode = 2;
  constant Integer TCP.TCPState_congestion_avoidance = 3;
  constant Integer TCP.TCPState_timeOut = 4;
  parameter Integer TCP.a = 1;
  parameter Integer TCP.L = 500;
  parameter Real TCP.timeout_th = 4;
  parameter Real TCP.m = 2.0;
  parameter Real TCP.RTO = 0.2;
  parameter Real TCP.w_max = 32.0;
  parameter Real TCP.MinSS = 1.0;
  discrete Integer TCP.state(start = 1);
  Real TCP.ssth(start = 65535.0 / /*Real*/(TCP.L)) "Slow
start thresholds";
  Real TCP.w(min = 1.0, max = TCP.w_max, start = TCP.MinSS);
  Real TCP.drop_timer(start = 0.0);
  Real TCP.retr_timer(start = 0.0);
  Boolean TCP.DDD(start = false);
  Real receiver.i.rate;
  Boolean receiver.i.drop;
  Real receiver.i.RTT(start = 0.02);
  parameter Boolean receiver.drop = false;
  parameter Real receiver.RTT = 0.0;
  Real receiver.received(start = 0.0);
  Real receiver.received_tot;
  Real receiver.r_tot_in;
  Real red_queue.i.rate;
```

```
  Boolean red_queue.i.drop;
  Real red_queue.i.RTT(start = 0.02;
  Real red_queue.o.rate;
  Boolean red_queue.o.drop;
  Real red_queue.o.RTT(start = 0.02);
  parameter Real red_queue.B_tot = 1250.;
  parameter Real red_queue.q_max = 300.0;
  parameter Integer red_queue.L = 500;
  parameter Real red_queue.T = 0.6;
  parameter Real red_queue.eps = 0.01;
  parameter Real red_queue.loss_factor_good = 2.0;
  parameter Real red_queue.loss_factor_bad = 1.0;
  parameter Real red_queue.lg = 8.0;
  parameter Real red_queue.lb = 4.0;
  parameter Real red_queue.my_random = 0.5;
  Real red_queue.q_tot(start = 0.0);
  Real red_queue.s_tot;
  Real red_queue.r_tot;
  Real red_queue.s_tot_acc(start = 0.0);
  Boolean red_queue.q_full(start = false);
  Integer red_queue.u_flow(start = 1);
  Integer red_queue.o_flow(start = 0);
  Real red_queue.q(start = 0.0);
  Real red_queue.rand;
  Real red_queue.z(start = 0.0) "Drop detection, bytes";
  Real red_queue.s;
  Real red_queue.r;
  Real red_queue.r_acc(start = 0.0);
  Boolean red_queue.loss;
  Real red_queue.rand_loss(start = 0.5);
  Boolean red_queue.GOOD_STATE;
  Boolean red_queue.BAD_STATE ;
  Real red_queue.g_tim(start = 1.0);
  Real red_queue.b_tim(start = -1.0);
  Real red_queue.loss_factor(start = 0.001);
  parameter Real link1.delay = 0.02 "link delay";
  Real link1.i.rate;
  Boolean link1.i.drop;
  Real link1.i.RTT(start = 0.02) "Round Trip Time for flow";
  Real link1.o.rate;
  Boolean link1.o.drop ;
  Real link1.o.RTT(start = 0.02);
  parameter Real link2.delay = 0.02 "link delay";
  Real link2.i.rate;
  Boolean link2.i.drop;
  Real link2.i.RTT(start = 0.02);
  Real link2.o.rate ;
  Boolean link2.o.drop ;
  Real link2.o.RTT(start = 0.02);
equation
  router1.i.rate = router1.o.rate;
  router1.i.RTT = router1.o.RTT;
  router1.i.drop = router1.o.drop;
```

```
        router2.i.rate = router2.o.rate;
        router2.i.RTT = router2.o.RTT;
        router2.i.drop = router2.o.drop;
        when (pre(TCP.state) == 1 or pre(TCP.state) == 3) and
TCP.state == 2 then
          reinit(TCP.retr_timer, TCP.o.RTT);
          reinit(TCP.ssth, 0.5 * TCP.w);
          reinit(TCP.w, 0.5 * TCP.w);
        end when;
        when (pre(TCP.state) == 1 or pre(TCP.state) == 3) and
TCP.state == 4 then
          reinit(TCP.retr_timer, TCP.RTO);
          reinit(TCP.ssth, (-0.5) * TCP.w);
          reinit(TCP.w, 1.0);
        end when;
        if TCP.state == 3 then
          der(TCP.w) = /*Real*/(TCP.a) / TCP.o.RTT;
          TCP.o.rate = TCP.w * /*Real*/(TCP.L) / TCP.o.RTT;
          der(TCP.retr_timer) = 0.0;
        elseif TCP.state == 2 then
          der(TCP.w) = 0.0;
          TCP.o.rate = TCP.w * /*Real*/(TCP.L) / TCP.o.RTT;
          der(TCP.retr_timer) = -1.0;
        elseif TCP.state == 4 then
          der(TCP.w) = 0.0;
          TCP.o.rate = 0.001;
          der(TCP.retr_timer) = -1.0;
        else
          der(TCP.w) = log(TCP.m) * TCP.w / TCP.o.RTT;
          TCP.o.rate = TCP.w * /*Real*/(TCP.L) / TCP.o.RTT;
          der(TCP.retr_timer) = 0.0;
        end if;
        der(TCP.drop_timer) = if TCP.drop_timer > 0.0 then -1.0
else 0.0;
        TCP.DDD = TCP.drop_timer < 0.0;
        when TCP.o.drop and TCP.drop_timer <= 0.0 then
          reinit(TCP.drop_timer, TCP.o.RTT);
        end when;
        der(TCP.ssth) = 0.0;
        der(receiver.received) = receiver.i.rate;
        receiver.r_tot_in = receiver.i.rate;
        receiver.received_tot = receiver.received;
        receiver.i.drop = receiver.drop;
        receiver.i.RTT = receiver.RTT;
        red_queue.s = red_queue.i.rate;
        red_queue.o.rate = (1.0 - red_queue.loss_factor) *
red_queue.r;
        red_queue.s_tot = red_queue.s;
        red_queue.r_tot = red_queue.r;
        red_queue.q_tot = red_queue.q;
        der(red_queue.s_tot_acc) = red_queue.s_tot;
        red_queue.o_flow = if red_queue.s_tot > red_queue.B_tot
then 1 else 0;
```

```
      red_queue.u_flow = 1 - red_queue.o_flow;
      red_queue.q_full = red_queue.q_tot > red_queue.q_max;
      red_queue.r = if red_queue.q_tot > red_queue.eps then
red_queue.B_tot * red_queue.q / red_queue.q_tot else
/*Real*/(red_queue.o_flow) * red_queue.B_tot * red_queue.s /
red_queue.s_tot + /*Real*/(red_queue.u_flow) * red_queue.s;
      der(red_queue.q) = if red_queue.q_tot < red_queue.q_max
then red_queue.s - red_queue.r else /*Real*/(red_queue.o_flow) *
(red_queue.s - red_queue.q * red_queue.s_tot / red_queue.q_tot)
+ /*Real*/(red_queue.u_flow) * (red_queue.s - red_queue.r);
      when edge(red_queue.GOOD_STATE) then
        reinit(red_queue.g_tim, (-red_queue.lg) *
log(red_queue.my_random));
      end when;
      when edge(red_queue.BAD_STATE) then
        reinit(red_queue.b_tim, (-red_queue.lb) *
log(red_queue.my_random));
      end when;
      red_queue.GOOD_STATE = red_queue.b_tim < 0.0;
      red_queue.BAD_STATE = red_queue.g_tim < 0.0;
      der(red_queue.g_tim) = if red_queue.g_tim > 0.0 then -1.0
else 0.0;
      der(red_queue.b_tim) = if red_queue.b_tim > 0.0 then -1.0
else 0.0;
      red_queue.loss_factor = if red_queue.GOOD_STATE then
red_queue.loss_factor_good else red_queue.loss_factor_bad;
      der(red_queue.r_acc) = red_queue.r;
      der(red_queue.rand_loss) = 0.0;
      red_queue.loss = red_queue.r_acc > 2.0 *
red_queue.rand_loss * /*Real*/(red_queue.L) /
red_queue.loss_factor;
      when red_queue.r_acc > 2.0 * red_queue.rand_loss *
/*Real*/(red_queue.L) / red_queue.loss_factor then
        reinit(red_queue.r_acc, 0.0);
      end when;
      when pre(red_queue.loss) then
        reinit(red_queue.rand_loss, red_queue.rand);
      end when;
      der(red_queue.z) = if red_queue.q_full then
red_queue.s_tot - red_queue.B_tot else 0.0;
      red_queue.rand = red_queue.my_random;
      if red_queue.z > /*Real*/(red_queue.L) and red_queue.q_tot
> red_queue.eps or edge(red_queue.q_full) then
        red_queue.i.drop = if 0.0 <= red_queue.rand and
red_queue.rand < red_queue.q / red_queue.q_tot then true else
red_queue.o.drop;
      else
        red_queue.i.drop = red_queue.o.drop or red_queue.loss;
      end if;
      red_queue.i.RTT = if red_queue.s <> 0.0 then red_queue.T +
red_queue.q_tot / red_queue.B_tot + red_queue.o.RTT else
red_queue.T + red_queue.o.RTT;
      link1.i.rate = link1.o.rate;
```

```
      link1.i.RTT = link1.o.RTT + 2.0 * link1.delay;
      link1.i.drop = link1.o.drop;
      link2.i.rate = link2.o.rate;
      link2.i.RTT = link2.o.RTT + 2.0 * link2.delay;
      link2.i.drop = link2.o.drop;
      TCP.o.RTT = link1.i.RTT;
      TCP.o.drop = link1.i.drop;
      TCP.o.rate = link1.i.rate;
      link1.o.RTT = router1.i.RTT;
      link1.o.drop = router1.i.drop;
      link1.o.rate = router1.i.rate;
      red_queue.i.RTT = router1.o.RTT;
      red_queue.i.drop = router1.o.drop;
      red_queue.i.rate = router1.o.rate;
      red_queue.o.RTT = router2.i.RTT;
      red_queue.o.drop = router2.i.drop;
      red_queue.o.rate = router2.i.rate;
      link2.i.RTT = router2.o.RTT;
      link2.i.drop = router2.o.drop;
      link2.i.rate = router2.o.rate;
      link2.o.RTT = receiver.i.RTT;
      link2.o.drop = receiver.i.drop;
      link2.o.rate = receiver.i.rate;
    algorithm
      TCP.state := 1;
      when edge(TCP.DDD) and TCP.w >= TCP.timeout_th and
(TCP.state == 1 or TCP.state == 3) then
        TCP.state := 2;
      elsewhen TCP.w >= TCP.ssth and TCP.state == 1 then
        TCP.state := 3;
      elsewhen TCP.w < TCP.timeout_th and edge(TCP.DDD) and
(TCP.state == 1 or TCP.state == 3) then
        TCP.state := 4;
      elsewhen TCP.retr_timer < 0.0 and TCP.state == 2 then
        TCP.state := 3;
      elsewhen TCP.retr_timer < 0.0 and TCP.state == 4 then
        TCP.state := 1;
      end when;
    end RED_simulation;
```