**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**
**РОССИЙСКОЙ ФЕДЕРАЦИИ**

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

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

«Допустить к защите»

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

# Выпускная квалификационная работа бакалавра

Направление 09.03.03 «Прикладная информатика»

Тема «Моделирование систем управления трафиком»

Выполнил студент Саргсян Арам Грачьяевич

Группа НПИбд-01-20
Студенческий билет № 1032201740

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

_____

Автор _____

**Москва**
**2024**

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

# Аннотация

**выпускной квалификационной работы**

Саргсяна Арама Грачьяевича

на тему: Моделирование систем управления трафиком

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

Автор ВКР _____ Саргсян Арам Грачьяевич

# Содержание

# Список используемых сокращений

NS — Network Simulator

NAM — Network Animator

RED — Random Early Detection

RED-QL — Random early detection-quadratic linear

GRED — Gentle Random Early Detection

ARED — Adaptive Random Early Detection

NLRED — Nonlinear RED

HRED — Hyberbola RED

TRED — Three section RED

RARED — Refined ARED

FARED — Fast ARED

DSRED — Double Slope RED

SmRED — Smart RED

TCL — Tool Command Language

TCP — Transmission Control Protocol

QoS — Quality of Service

FTP — File Transport Protocol

AIMD — additive-increase/multiplicative-decrease

# Введение

Данное исследование посвящено анализу и сравнению алгоритмов управления очередью из семейства RED, реализованных с использованием программных средств NS-2 и Mininet. Центральная задача работы - исследовать принципы работы и эффективность алгоритмов RED через моделирование их поведения в различных сетевых условиях. В рамках работы предпринята попытка имтационного и натурного моделирования поведения сети при использовании алгоритмов RED, а также проведено сравнение их производительности с целью выявления наиболее эффективных настроек для обеспечения качественной и надежной передачи данных. Результаты моделирования предназначены для определения оптимальных параметров алгоритмов управления очередью, способствующих улучшению общей производительности сетевых систем.

## Актуальность темы

Актуальность данного исследования заключается в анализе механизма активного управления очередью RED, который вносит вклад в оптимизацию распределения ресурсов сети и обеспечивает соответствие требованиям пользователей по скорости и надежности передачи данных.

## Цель работы:

Целью данной выпускной квалификационной работы является исследование и анализ алгоритмов семейства RED для активного управления очередью в сетевых системах, а также оценка их эффективности с использованием различных инструментов моделирования

## Структура работы

Данная работа состоит из введения, трех разделов, заключения, списка используемой литературы и приложением. Во введении мною приведено краткое описание работы, также обусловлена ее актуальность, поставлена цель и сформулированы задачи выпускной квалификационной работы.

В первом разделе работы рассмотрены средства моделирования сетей и принципы их работы.

Во втором разделе работы приведен обзор алгоритмов семейства RED и её реализации в итационной модели NS-2 и натурной модели в Mininet.

В третьем разделе выпускной квалификационной работы подробно описаны функции, отвечающие за реализацию моделей. Показаны результаты моделирования, выведены

необходимые графики функций и сделаны выводы об эффективности алгоритма с помощью двух средств моделирования.

В заключении подведены общие итоги работы, изложены основные выводы.

# Глава 1. Средства моделирования сетей передачи данных

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

## 1.1. NS-2

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

NS-2 разработан на языке программирования C++ и TCL, имеет открытый исходный код, что обеспечивает гибкость и расширяемость средства моделирования, например самостоятельную реализацию алгоритма новых дисциплин управления очередью. NS-2 содержит библиотеку классов, которые представляют различные элементы сети, такие как узлы, маршрутизаторы, каналы связи и протоколы передачи данных. Для создания модели сети определяются характеристики и параметры каждого элемента сети: пропускная способность канала, задержки, вероятность потери пакетов и другие. После завершения симуляции NS-2 предоставляет мощные инструменты анализа результатов, включая возможность визуализации данных посредством программы NAM, статистический анализ и сравнение результатов экспериментов, что позволяет изучать и оценивать производительность различных протоколов и алгоритмов в различных сценариях сети [12; 17].

Алгоритм работы в NS-2 включает в себя несколько ключевых шагов:

1. **Формирование структуры сетевого взаимодействия:** процесс конструирования структуры сетевого взаимодействия включающая в себя разработку схемы, иллюстрирующей взаимосвязь между узлами сети, а также определение компонентов сети, их связей, а также источников и приемников информационных потоков.

2. **Конфигурация параметров для имитационного моделирования:** настройка параметров моделирования, к которым относятся настройка сетевых протоколов, установка размеров буферов, временных задержек, пропускной способности каналов связи и других сетевых характеристик.

3. **Разработка сценария имитационного моделирования:** создание сценария, описывающего последовательность событий в сети, включая передачу данных между узлами, модификации в структуре сети, адаптацию параметров сетевых протоколов и прочее.

4. **Инициация процесса моделирования:** запуск процесса моделирования, в ходе которого NS-2 выполняет воспроизведение передачи данных через сеть, реагируя на события согласно установленным параметрам и сценарию.

5. **Сбор данных и их анализ по завершении моделирования:** В процессе имитационного моделирования NS-2 аккумулирует информацию о показателях работы сети, включая задержки, пропускную способность, частоту потерь пакетов и другие важные метрики.

## 1.2. Mininet

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

Алгоритм работы в Mininet включает в себя несколько ключевых шагов:

1. **Подготовка виртуальной сетевой инфраструктуры:** создание виртуальной сетевой топологии с использованием пространств имен Linux и технологии виртуализации, позволяющее моделировать работу отдельных сетевых узлов, как коммутаторов, так и конечных устройств, связывая их виртуальными каналами связи с заданными параметрами пропускной способности, задержки и потерь.

2. **Настройка параметров сетевой топологии:** конфигурация созданной виртуальной сети, включая назначение IP-адресов узлам, определение маршрутов передачи данных и установление правил обработки трафика на коммутаторах.

3. **Эмуляция сетевых условий:** имитация различных условий работы сети, таких как изменение пропускной способности каналов, введение искусственных задержек и эмуляция потерь пакетов, что позволяет оценить поведение сетевых протоколов и приложений в разнообразных сценариях, включая условия высокой загруженности сети и ненадежности каналов связи.

4. **Запуск и мониторинг симуляции:** запуск симуляции в Mininet, в процессе которого можно осуществлять мониторинг состояния сети, отслеживая ключевые метрики производительности.

5. **Анализ полученных данных:** завершающий этап работы с Mininet включающая в себя сбор и анализ данных, полученных в ходе симуляции, что позволяет оценить эффективность сетевых протоколов, алгоритмов маршрутизации и стратегий управления трафиком, а также верифицировать теоретические модели работы сети на практике.

# Глава 2. Обзор дисциплины управлением очередью RED

В данном разделе представим описание работы нескольких алгоритмов семейства RED и их реализацию в NS-2 и mininet.

## 2.1. Классическая модификация

RED — это семейство механизмов предотвращения перегрузки на шлюзе. Он основан на общих принципах, полезен для управления средним размером очереди в сети, где не доверяют взаимодействию между протоколами передачи данных. В контрасте с подходом Droptail, который предусматривает простое отбрасывание входящих пакетов при достижении максимальной емкости очереди, RED учитывает потоки трафика в сети и стремится предоставить равную пропускную способность для каждого соединения, что позволяет избежать перегрузки сети и улучшить качество обслуживания. В оригинальном RED маршрутизатор вычисляет усредненный по времени средний размер очереди с использованием фильтра нижних частот (экспоненциально взвешенное скользящее среднее) или сглаживания по длине выборки очередей, средний размер очереди сравнивается с двумя пороговыми значениями: минимальным порогом и максимальным. Когда средний размер очереди меньше минимального порога, пакеты не отбрасываются, когда средний размер очереди превышает максимальный порог, отбрасывается все поступающие пакеты. Если размер средней очереди находится между минимальным и максимальным порогом, пакеты отбрасываются с вероятностью $p$, которая линейно увеличивается до тех пор, пока средняя очередь не достигнет максимального порога. Подробно классический алгоритм описан в [4; 20].

На рисунке 2.1 представлена вероятная функция сброса пакетов.

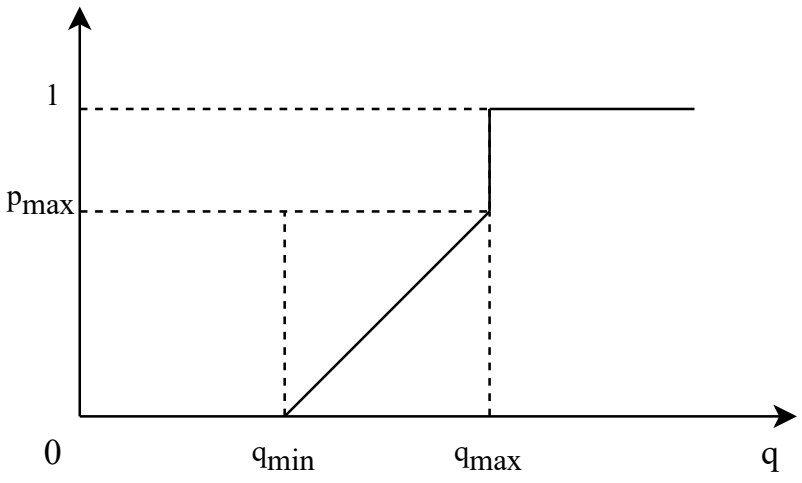


**Рис. 2.1.. Вид функции сброса в алгоритме RED**

Вероятность $p_b$ маркировки на отбрасывание пакетов представляет собой функцию, линейно зависящую от $\hat{q}$ (средневзвешенное скользящее среднее), минимального $q_{\min}$ и максимального $q_{\max}$ пороговых значений и параметра $p_{\max}$, определяющего часть отбрасываемых пакетов при достижении средним размером очереди значения $q_{\max}$ и вычисляется следующим образом (2.1):

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

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

`double REDQueue::calculate_p_ne файла red.cc`

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

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

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

интерфейса. Не влияет на параметр пропускной скорости интерфейса. Необязательное значение.

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

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

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

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

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

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

## 2.2. Нелинейные модификации

### 2.2.1. NLRED

Nonlinear RED — это модификация классического алгоритма RED, в котором используется нелинейная функция для определения вероятности отбрасывания пакетов. Nonlinear RED предназначен для более точной адаптации к изменениям трафика и динамике сети. Он способен эффективно реагировать на изменения величины очереди и адаптироваться к различным условиям сети. Это позволяет более гибко управлять задержкой пакетов и предотвращать перегрузки в сети, что делает Nonlinear RED более эффективным по сравнению с классическим алгоритмом RED. [10; 19].

Вероятность $p_b$ маркировки на отбрасывание пакетов вычисляется следующим способом (2.2):

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\text{min}}, \\ 1.5(\frac{\hat{q}-q_{\text{min}}}{q_{\text{max}}-q_{\text{min}}})^2 p_{\text{max}}, & q_{\text{min}} < \hat{q} \leqslant q_{\text{max}}, \\ 1, & \hat{q} > q_{\text{max}}. \end{cases} \tag{2.2}$$

По умолчанию NLRED не реализован в NS-2. Для её добавления я использовал патч для данной модификации, созданный Mohit P. Tahiliani для версии 2.34, совместимой также для версии 2.35 [14].

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

### 2.2.2. GRED

GRED (Gentle Random Early Detection, мягкое/аккуратное произвольное раннее обнаружение) — алгоритм активного управления очередью, является расширением RED. Стандартный алгоритм увеличивает вероятность отбрасывания с 0.05 до 0.5, когда средняя длина очереди увеличивается от минимального до максимального порогового значения, но при превышении максимального порога вероятность возрастает напрямую с 0.5 до 1. Этот внезапный скачок нормализуется модификацией Gentle RED, который расширяет RED тем, что добавляет дополнительное максимальное пороговое значние, которое равно $2q_{\text{max}}$, тем самым «сглаживая» кривую [13]. Однако, например, задача минимального порога в данной модификации не меняется, и увеличение лишь максимального порога для отбрасывания всех пакетов делает GRED лишь частным случаем классического алгоритма. Данная модификация в NS-2 используется по умолчанию, так как переменная `gentle_` по умолчанию является истинной.

Вероятность сброса определяется по формуле (2.3):

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\min}, \\ \frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}} p_{\max}, & q_{\min} \leqslant \hat{q} < q_{\max}, \\ \frac{\hat{q}-q_{\min}}{q_{\max}}(1-p_{\max}) - p_{\max}, & q_{\max} \leqslant \hat{q} < 2q_{\max}, \\ 1, & \hat{q} \geqslant q_{\max}. \end{cases} \tag{2.3}$$

### 2.2.3. DS-RED

Алгоритм DS-RED — это ещё одна модификация RED, в котором вводится дополнительное пороговое значение $q_{mid}$ между минимальным $q_{min}$ и максимальным REDФункция сброса описывается двумя линейными сегментами с углами наклона $\alpha$ и $\beta$ соответственно, регулируемыми задаваемым селектором режимов $\gamma$ [18].

Функция вероятности сброса пакетов в алгоритме DSRED показана в формуле (2.4)

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{min}, \\ \alpha\hat{q} - q_{min}, & q_{min} \leqslant \hat{q} < q_{mid}, \\ 1 - \gamma + \beta\hat{q} - q_{mid}, & q_{mid} \leqslant \hat{q} < q_{max}, \\ 1, & \hat{q} \geqslant q_{max}. \end{cases} \tag{2.4}$$

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

Режим работы DSRED настраивается с помощью изменения наклона функции выброса DSRED с использованием параметра $\gamma$. Регулируя только $\gamma$, DSRED способен достичь высокой скорости выброса, за которой следует низкая скорость выброса, или наоборот, обеспечивая гибкую схему управления (AQM) для решения сложных ситуаций с сетевой конгестией.

По умолчанию модификация не реализована в NS-2, для её реализации дополнил функцию `double REDQueue::calculate_p_ne файла red.cc`, а в программе очереди указал значение переменной `double_slope_ 1`.

### 2.2.4. HRED

HRED —это модификация классического RED с нилейно возрастающей функцией отбрасывания пакетов в сети. Использование HRED гиперболы в качестве кривой вероятности выброса может регулировать размер очереди к $q_{\max}$ в гораздо более широком диапазоне нагрузок на трафик. Другими словами, HRED нечувствителен к уровню сетевой нагрузки, в результате чего задержка в очереди становится более предсказуемой, поскольку размер очереди не изменяется сильно в зависимости от уровня конгестии.

HRED сохраняет способность контролировать кратковременную перегрузку путем поглощения пакетных потоков, так как он все еще использует алгоритм подсчета среднего размера очереди и поддерживает неполную очередь. HRED прост в реализации и легко внедряется на маршрутизаторах, так как зменяется только профиль отбрасывания по сравнению с классическим алгоритмом RED [5]. Для его реализации дополнил фунцию `double REDQueue::calculate_p_ne файла red.cc`, а в программе очереди указал значение переменной `hyperbola_ 1`.

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

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

### 2.2.5. TRED

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

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

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

### 2.2.6. RED-QL

RED-QL —модификация алгоритма RED, также является разновидностью алгоритма с нелинейно возрастающей функцией. RED-QL имеет квадратично-линейную форму и определяется на основе параметров, которые могут быть настроены для определенных требований сети[9] По умолчанию модификация не реализована в NS-2, для её реализации дополнил фунцию `double REDQueue::calculate_p_ne файла red.cc`, а в программе очереди указал значение переменной `quadratic_linear_ 1.`.

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

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

## 2.3. Адаптивные модификации

### 2.3.1. ARED

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

Алгоритм ARED функционирует следующим образом ((2.8)), ((2.9)). Для каждого интервала `interval` (параметр) в секундах, если $\hat{q}$ больше целевой (желаемой) $\hat{q_t}$ и $p_{\max} \leqslant 0,5$, то $p_{\max}$ увеличивается на некоторую величину $\alpha$; в противном случае, если $\hat{q}$ меньше целевой

$\hat{q_t}$ и $p_{\max} \geqslant 0,01$, то $p_{\max}$ уменьшается в $\beta$ раз, $\alpha$ и $\beta$ задаются командами `set alpha_` и `set beta_`:

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

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

Основные особенности:

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

### 2.3.2. SmRED

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

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

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

### 2.3.3. RARED

Алгоритм RARED предлагает более активно изменять вероятность сброса $p_{\max}$, чтобы иметь возможность быстрой адаптации к изменяющейся экспоненциально взвешенной скользящей средней длине очереди $\hat{q}$ [8].

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

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

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

По умолчанию RARED не реализован в NS-2. Для её добавления я использовал патч для данной модификации, созданный Mohit P. Tahiliani для версии 2.34, совместимой также для версии 2.35 [15].

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

### 2.3.4. POWARED

Powared является модификацией алгоритма ARED [11]. В данной модификации величина $p_{max}$ максимального сброса считается следующим образом (2.13). Алгоритм POWARED более агрессивно реагирует на изменение средней очереди, чем ARED. Данная модификация не реализована в NS-2, для её моделирования я в файл red.cc добавил функцию `void REDQueue::updateMaxP_powared`, а в настройке очереди сети указал значение переменных `adaptive_ 1` и `powared_ 1`. Параметры модификации задаются с помощью переменных `pwk_`, `pwb_`.

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

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

### 2.3.5. FARED

FARED — это алгортитм, который сохраняет целевой диапазон, указанный в алгоритме RARED, но изменяет верхнюю и нижнюю границы для $\alpha$ и $\beta$ соответственно. Алгоритм FARED обеспечивает надежную производительность в широком диапазоне сред, включая сценарии с умеренной и высокой нагрузкой на трафик [16]. Данная модификация не требует установки каких-либо дополнительных параметров для повышения производительности. Поскольку в алгоритм FARED внесены лишь незначительные изменения по сравнению с ARED и ReARED, он может быть развернут без каких-либо сложностей (2.14), (2.15). Данная модификация не реализована в NS-2, для её моделирования я в файл red.cc добавил функцию `void REDQueue::updateMaxP_fast_adaptive`, а в настройке очереди сети указал значение переменных `adaptive_ 1` и `fast_adaptive_ 1`.

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

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

# Глава 3. Моделирование систем с алгоритмом RED в NS-2 и Mininet

В данном разделе представлены результаты исследований. Для запуска моделей используем сеть с топологией, представленной в  3.1.

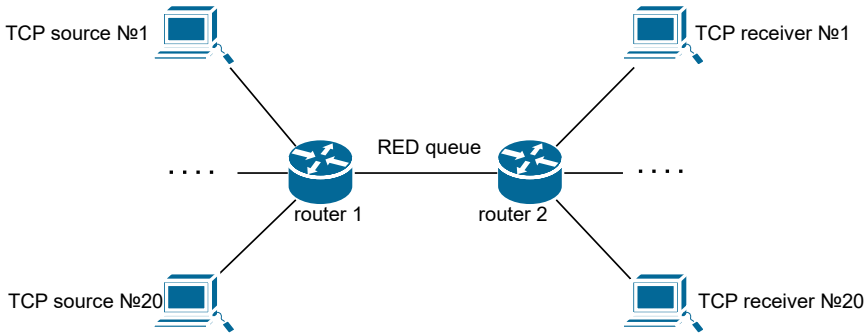


**Рис. 3.1.. Схема топологии моделируемой сети**

Для данной топологии задали следующие параметры:

- между TCP-источниками и первым маршрутизатором дуплексные соединения с пропускной способностью 100 Мбит/с и задержкой 20 мс очередью типа DropTail;
- между TCP-приёмниками и вторым маршрутизатором установлены дуплексные соединения с пропускной способностью 100 Мбит/с и задержкой 20 мс очередью типа DropTail;
- между маршрутизаторами установлено симплексное соединение ($R1–R2$) с пропускной способностью 20 Мбит/с и задержкой 15 мс очередью типа RED, размером буфера 300 пакетов; в обратную сторону — симплексное соединение ($R2–R1$) с пропускной способностью 15 Мбит/с и задержкой 20 мс очередью типа DropTail;
- параметры алгоритма RED: $q_{\min} = 75, q_{\max} = 150, q_w = 0,002, p_{\max} = 0.1$;
- максимальный размер TCP-окна 32; размер передаваемого пакета 1000 байт; байт; время моделирования —100 единиц модельного времени.

## 3.1. Моделирование сети в NS-2

В имитационной модели TCP-приемники/источники, а также маршрутизаторы реализованы как стандартные узлы, данные передаются по протоколу FTP поверх TCP-Reno,

для получения данных используются стандартные в программе средства, графики визуализируются в xgraph(для быстрого просмотра) и в GNUPLOT(для дальнейшего анализа). Модифицированный NS-2 представлен в репозитории [21]:

Запустив первую модель с заданными параметрами, получили следующие результаты 3.2.

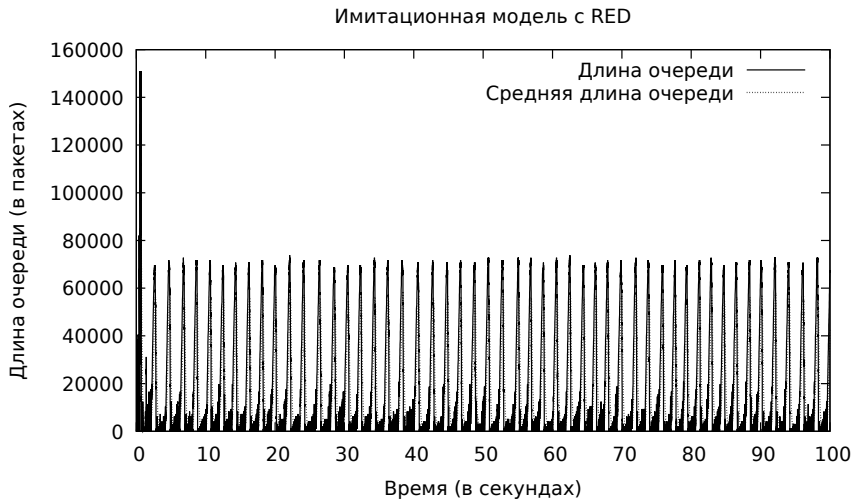


**Рис. 3.2.. График длины очереди и средней длины очереди на линке между маршрутизаторами**

## 3.2. Моделирование сети в Mininet

Модель в Mininet более сложная в реализации. В нем используются реальные виртуальные маршрутизаторы из класса LinuxRouter, использованы коммутаторы для подсоединения к маршрутизаторам нескольких оконечных устройств, а также настроена ip-адресация для корректной работы программы. Для настройки очереди использован tc в которой реализованы только модийикации RED и ARED, передача пакетов происходит с помощью iperf3 и выводятся в json файл, из которой данные извлекаются с помощью shell скрипта из [7]. Для запуска на нескольких устройствах используются паралельные потоки. Для визуализации также используется GNUPLOT.

## 3.3. Сравнение результатов в 2 средствах моделирования

Текст.

# Заключение

В ходе выполнения выпускной квалификационной работы был дан обзор используемых средств моделирования, типов дисциплины управлением очередью RED, а также рассмотрен алгоритм работы дисциплин данного семейства. Разработан единый программный модуль, реализующий алгоритм работы RED в NS-2 и Mininet. Были задействованы стандартные классы, а также средство Mininet, позволяющее строить графики по полученным данным. Таким образом были выполнены задачи, сформулированные перед началом работы

В работе было рассмотрено:

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

# Список литературы


1. A self-configuring RED gateway / W.-C. Feng [и др.] // IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320). — IEEE, 1999. — DOI: 10.1109/infcom.1999.752150.
2. An AQM based congestion control for eNB RLC in 4G/LTE network / A. K. Paul [и др.] // 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE). — IEEE, 05.2016. — DOI: 10.1109/ccece.2016.7726792.
3. Congestion Control Scheme Performance Analysis Based on Nonlinear RED / C.-W. Feng [и др.] // IEEE Systems Journal. — 2017. — Дек. — Т. 11, № 4. — С. 2247—2254. — DOI: 10.1109/jsyst.2014.2375314.
4. *Floyd S.*, *Jacobson V.* Random early detection gateways for congestion avoidance // IEEE/ACM Transactions on Networking. — 1993. — Т. 1, № 4. — С. 397—413. — DOI: 10.1109/90.251892.
5. *Hu L.*, *Kshemkalyani A.* HRED: a simple and efficient active queue management algorithm // Proceedings. 13th International Conference on Computer Communications and Networks (IEEE Cat. No.04EX969). — IEEE, 2004. — DOI: 10.1109/icccn.2004.1401681.
6. *Keti F.*, *Askar S.* Emulation of Software Defined Networks Using Mininet in Different Simulation Environments // 2015 6th International Conference on Intelligent Systems, Modelling and Simulation. — IEEE, 02.2015. — DOI: 10.1109/isms.2015.46.
7. *Kfoury E.* Shell script that plots Iperf3's JSON file. — 2021. — URL: https://github.com/ekfoury/iperf3_plotter.
8. *Kim T.-H.*, *Lee K.-H.* Refined Adaptive RED in TCP/IP Networks // SICE-ICASE International Joint Conference 2006 Oct. 18-2 1, 2006 in Bexco, Busan, Korea. — IEEE, 2006. — DOI: 10.1109/SICE.2006.314633.
9. *Kumhar D.*, *kumar A.*, *Kewat A.* QRED: an enhancement approach for congestion control in network communications // International Journal of Information Technology. — 2020. — Окт. — Т. 13, № 1. — С. 221—227. — DOI: 10.1007/s41870-020-00538-1.
10. Nonlinear AQM for Multiple RED Routers / L. Lu [и др.] // 2008 Third International Conference on Convergence and Hybrid Information Technology. — IEEE, 11.2008. — DOI: 10.1109/iccit.2008.68.
11. POWARED for Non-Linear Adaptive RED / B. Ng [и др.] // 2005 Asia-Pacific Conference on Communications. — IEEE, 2005. — DOI: 10.1109/apcc.2005.1554179.

12. *Rehmani M. H.*, *Saleem Y.* Network Simulator NS-2 // Encyclopedia of Information Science and Technology, Third Edition. — IGI Global, 07.2014. — С. 6249—6258. — DOI: 10.4018/978-1-4666-5888-2.ch615.

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

14. *Tahiliani M. P.* Nonlinear RED (NLRED) patch for NS-2. — 01.2012. — URL: https://mohittahiliani.blogspot.com/2012/01/nonlinear-red-nlred-patch-for-ns-2.html.

15. *Tahiliani M. P.* Refined Adaptive RED (Re-ARED or RARED) patch for NS-2. — 01.2012. — URL: https://mohittahiliani.blogspot.com/2012/01/refined-adaptive-red-re-ared-or-rared.html.

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

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

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

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

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

21. *Саргсян А. Г.* Реализация модификаций RED в NS-2. — 2024. — URL: https://github.com/agsargsyan/ns2_with_red.

# Приложение А. Имитационная модель

## main.tcl

```
#новый экземпляр объекта Symulator
set ns [new Simulator]
#трейс файл для nam
set nf [open output/out.nam w]
$ns namtrace-all $nf
#количество источников
set N 20
#создание узлов и соединений
source "nodes.tcl"
#метрики TCP
source "TCP.tcl"
#настройка очереди
source "queue.tcl"
#настройка времени моделирования
source "timing.tcl"
#визуализация в nam
source "nam.tcl"
#процедура finish
source "finish.tcl"
#запуск программы
$ns run
```

## nodes.tcl

```
#маршрутизаторы
set node_(r0) [$ns node]
set node_(r1) [$ns node]

#источники и приемники
for {set i 0} {$i < $N} {incr i} {
set node_(s$i) [$ns node]
set node_(s[expr $N + $i]) [$ns node]
}

#связи между маршрутизаторами и другими узлами(размер буфера, время, тип очере
for {set i 0} {$i < $N} {incr i} {
$ns duplex-link $node_(s$i) $node_(r0) 100Mb 20ms DropTail
$ns duplex-link $node_(s[expr $N + $i]) $node_(r1) 100Mb 20ms DropTail
}

#связи между маршрутизаторами(размер буфера, время, тип очереди)
$ns simplex-link $node_(r0) $node_(r1) 20Mb 15ms RED
```

```
$ns simplex-link $node_(r1) $node_(r0) 15Mb 20ms DropTail

# Агенты и приложения
for {set t 0} {$t < $N} {incr t} {
$ns color $t green
set tcp($t) [$ns create-connection TCP/Reno $node_(s$t) TCPSink $node_(s[expr
$tcp($t) set window_ 32
$tcp($t) set maxcwnd_ 32
set ftp($t) [$tcp($t) attach-source FTP]
}
```

## TCP.tcl

```
# Мониторинг метрик TCP

#размер окна TCP для всех источников
set windowVsTime [open output/WvsT w]

#размер окна TCP для 1 источника
set windowVsTime_1 [open output/WvsT_1 w]

#время приема-передачи
set rtt [open output/RTT w]

#отклонение времени приема-передачи
set rttvar [open output/RTTVAR w]

# Функция для получение данных о метриках
proc plotMetric {tcpSource file metric} {
    global ns
    set time 0.01
    set now [$ns now]
    set value [$tcpSource set $metric]
    puts $file "$now $value"
    $ns at [expr $now+$time] "plotMetric $tcpSource $file $metric"
}
```

## queue.tcl

```
#Лимит очереди
$ns queue-limit $node_(r0) $node_(r1) 300
$ns queue-limit $node_(r1) $node_(r0) 300

#настройка параметров RED
set redq [[$ns link $node_(r0) $node_(r1)] queue]
```

```
$redq set thresh_ 75
$redq set maxthresh_ 150
$redq set q_weight_ 0.002
$redq set linterm_ 10
$redq set drop-tail_ true
$redq set gentle_ false
$redq set queue-in-bytes false

#мониторинг параметров длины очереди:
set tchan_ [open output/all.q w]
$redq trace curq_
$redq trace ave_
$redq attach $tchan_

#монитринг соеденения между маршрутизаторами
set qmon [$ns monitor-queue $node_(r0) $node_(r1) [open output/qm.out w]]
[$ns link $node_(r0) $node_(r1)] queue-sample-timeout

#Для реализации разных модификаций RED, реализовано благодаря изменению исходно

#$redq set nonlinear_ 1
#$redq set hyperbola_ 1
#$redq set quadratic_linear_ 1
#$redq set three_sections_ 1
#$redq set exponential_ 1
#$redq set smart_ 1
#$redq set double_slope_ 1

#Группа адаптивных алгоритмов
#$redq set adaptive_ 1
#$redq set feng_adaptive_ 1
#$redq set refined_adaptive_ 1
#$redq set fast_adaptive_ 1
$redq set powared_ 1
```

## timing.tcl

```
for {set r 0} {$r < $N} {incr r} {
$ns at 0.0 "$ftp($r) start"
$ns at 100.0 "$ftp($r) stop"
$ns at 1.0 "plotMetric $tcp($r) $windowVsTime cwnd_"
}
$ns at 1.0 "plotMetric $tcp(1) $windowVsTime_1 cwnd_"
$ns at 1.0 "plotMetric $tcp(1) $rtt rtt_"
$ns at 1.0 "plotMetric $tcp(1) $rttvar rttvar_"
$ns at 100.0 "finish"
```

## nam.tcl

```
#визуализация цветов, формы, располажения узлов в nam
$node_(r0) color "red"
$node_(r1) color "red"
$node_(r0) label "RED"
$node_(r1) shape "square"
$node_(r0) label "square"

$ns simplex-link-op $node_(r0) $node_(r1) orient right
$ns simplex-link-op $node_(r1) $node_(r0) orient left
$ns simplex-link-op $node_(r0) $node_(r1) queuePos 0
$ns simplex-link-op $node_(r1) $node_(r0) queuePos 0

for {set m 0} {$m < $N} {incr m} {
$ns duplex-link-op $node_(s$m) $node_(r0) orient right
$ns duplex-link-op $node_(s[expr $N + $m]) $node_(r1) orient left
}

for {set i 0} {$i < $N} {incr i} {
$node_(s$i) color "blue"
$node_(s$i) label "ftp"

}
```

## finish.tcl

```
#Finish procedure
proc finish {} {
   global ns nf
   $ns flush-trace
   close $nf
   global tchan_
   #разделение данных мгновееной очереди и средней очереди
   set awkCode {
      {
 if ($1 == "Q" && NF>2) {
    print $2, $3 >> "output/temp.q";
    set end $2
 }
 else if ($1 == "a" && NF>2)
 print $2, $3 >> "output/temp.a";
      }
   }
   set f [open output/temp.queue w]
   puts $f "TitleText: RED"
```

```
    puts $f "Device: Postscript"

    if { [info exists tchan_] } {
       close $tchan_
    }
    exec rm -f output/temp.q output/temp.a
    exec touch output/temp.a output/temp.q
    exec awk $awkCode output/all.q

    puts $f \"queue
    exec cat output/temp.q >@ $f
    puts $f \n\"ave_queue
    exec cat output/temp.a >@ $f
    close $f
    # вывод графиков в xgraph для быстрого просмотра
    exec xgraph -bb -tk -x time -t "TCPRenoCWND" output/WvsT &
    exec xgraph -bb -tk -x time -t "TCPRenoCWND_1" output/WvsT_1 &
    exec xgraph -bb -tk -x time -t "RTT" output/RTT &
    exec xgraph -bb -tk -x time -t "RTTVAR" output/RTTVAR &
    exec xgraph -bb -tk -x time -y queue output/temp.queue &
    #exec nam output/out.nam &
    exit 0
}
```

# Приложение В. Реализация модификаций RED в ns-2

## В.1. Обновленная вероятностей функции RED в NS-2

```
double
REDQueue::calculate_p_new(double v_ave, double th_max, int gentle, double v_a,
double v_b, double v_c, double v_d, double max_p)
{
double target;
double exponenta = 2.7182818285;
double th_min = edp_.th_min;
double p;
if (gentle && v_ave >= th_max) {
// p ranges from max_p to 1 as the average queue
// size ranges from th_max to twice th_max
p = v_c * v_ave + v_d;
        } else if (!gentle && v_ave >= th_max) {
                // OLD: p continues to range linearly above max_p as
                // the average queue size ranges above th_max.
                // NEW: p is set to 1.0
                p = 1.0;
        } else if (edp_.quadratic_linear == 1) {
         target = 2 * ((th_min + th_max)/3) - th_min;
         if(v_ave < target){
         p = 9 * max_p * ((v_ave-th_min)/(2*(th_max-2*th_min))) * ((v_ave-
th_min)/(2*(th_max-2*th_min)));
         } else if (v_ave >= target) {
         p = max_p + 3*(1-max_p)*((v_ave-target)/(th_max+th_min));
         }
        } else if (edp_.improved == 1) {
         target = ((th_min + th_max)/3) + th_min;
         if(v_ave < target){
         p = 9 * max_p * ((v_ave-th_min)/(th_max + th_min)) * ((v_ave-th_min)/
         } else if (v_ave >= target) {
         p = max_p + 3*(1-max_p)*(v_ave-target)/(2*(th_max - 2 * th_min));
         }
        } else if (edp_.smart == 1) {
         target = ((th_max - th_min)/2) + th_min;
         if(v_ave < target){
         p = max_p * pow(((v_ave-th_min)/(th_max - th_min)), 2);
         } else if (v_ave >= target) {
         p = max_p * pow(((v_ave-th_min)/(th_max - th_min)), 0.5);
         }
        } else if (edp_.three_sections == 1){
         double delta = (th_min+th_max/3);
         if (v_ave < (th_min + delta)){
         p = 9 * max_p * pow((v_ave-th_min)/(th_max-th_min), 3) ;
```

```
         }
         else if ((v_ave >= th_min + delta) && (v_ave < th_min + 2 * delta)){
         p = max_p * (v_ave-th_min)/(th_max-th_min);
         }
         else if (v_ave >= th_min + 2* delta){
         p =  9 * max_p * pow((v_ave-th_min)/(th_max-th_min), 3) + max_p;
         }
        }
        else if (edp_.double_slope == 1) {
         double a = (2-2* edp_.omega)/(th_max - th_min);
         double b = (2 * edp_.omega)/(th_max - th_min);
         target = ((th_max + th_min)/2);
         if(v_ave < target){
         p = a * (v_ave-th_min);
         } else if (v_ave >= target) {
         p = 1 - edp_.omega + b * (v_ave - target);
         }
        }
        else {
                // p ranges from 0 to max_p as the average queue
                // size ranges from th_min to th_max
                p = v_a * v_ave + v_b;
                // p = (v_ave - th_min) / (th_max - th_min)

                /* Added by Mohit P. Tahiliani for Nonlinear RED (NLRED) - Sta
if(edp_.nonlinear == 1){
p *= p; // This ensures probability is a quadratic function of "average queue
}
else if (edp_.hyperbola == 1){
p *= 1/p; // This ensures probability is a hyperbola function of "average queu
}
else if (edp_.exponential == 1){ // Used for RED_e
p = (pow(exponenta, v_ave)-pow(exponenta, th_min))/(pow(exponenta, th_max)-
pow(exponenta, th_min));
}
                p *= max_p;
        }
if (p > 1.0)
p = 1.0;
return p;
}
```

## B.2. Функции для адаптивных RED

```
void REDQueue::updateMaxP_refined_adaptive(double new_ave, double now)
{
```

```
   double part = 0.48*(edp_.th_max - edp_.th_min);
  if ( new_ave < edp_.th_min + part && edv_.cur_max_p > edp_.bottom) {
  edv_.cur_max_p = edv_.cur_max_p * (1.0 - (0.17 * ((edp_.th_min + part) - new
  edv_.lastset = now;
  double maxp = edv_.cur_max_p;
  } else if (new_ave > edp_.th_max - part && edp_.top > edv_.cur_max_p ) {
  double alpha = edp_.alpha;
  alpha = 0.25 * edv_.cur_max_p * ((new_ave - (edp_.th_max - part)) / (edp_.th
  edv_.cur_max_p = edv_.cur_max_p + alpha;
  edv_.lastset = now;
  double maxp = edv_.cur_max_p;
  }
}

void REDQueue::updateMaxP_powared(double new_ave, double now)
{
   double target = 0.5*(edp_.th_max + edp_.th_min);
   int k = edp_.pwk;
   int b = edp_.pwb;
   int r = edp_.bf_size;
   double v_ave = edv_.v_ave;



   double delta1 = abs(pow((v_ave - target)/(b * target), k));
   double delta2 = abs(pow((target - v_ave)/(b * (r -target)), k));

  if ( new_ave < target && edv_.cur_max_p > edp_.bottom) {
  edv_.cur_max_p = edv_.cur_max_p - delta1;
  edv_.lastset = now;
  double maxp = edv_.cur_max_p;
  } else if (new_ave > target && edp_.top > edv_.cur_max_p ) {
  edv_.cur_max_p = edv_.cur_max_p + delta2;
  edv_.lastset = now;
  double maxp = edv_.cur_max_p;
  }
}

void REDQueue::updateMaxP_fast_adaptive(double new_ave, double now){
   double part = 0.48*(edp_.th_max - edp_.th_min);
if ( new_ave < edp_.th_min + part && edv_.cur_max_p > edp_.bottom) {
  edv_.cur_max_p = edv_.cur_max_p * (1.0 - (0.0385 * ((edp_.th_min + part) - n
  edv_.lastset = now;
  double maxp = edv_.cur_max_p;
  } else if (new_ave > edp_.th_max - part && edp_.top > edv_.cur_max_p ) {
  double alpha = edp_.alpha;
  alpha = 0.0412 * edv_.cur_max_p * (new_ave - part) / part;
```

```
  edv_.cur_max_p = edv_.cur_max_p + alpha;
  edv_.lastset = now;
  double maxp = edv_.cur_max_p;
  }
}
```

# Список иллюстраций

# Список таблиц