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 — Hyperbola 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

RTT — Round-Trip Time

UDP — User Datagram Protocol

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 реализация моделей с дисциплиной RED с использованием разных средств моделирования, а также анализ полученных результатов.

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. Обзор дисциплины управлением очередью RED

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

1.1. 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 [1; 2].

На рисунке 1.1 представлена вероятная функция сброса пакетов.

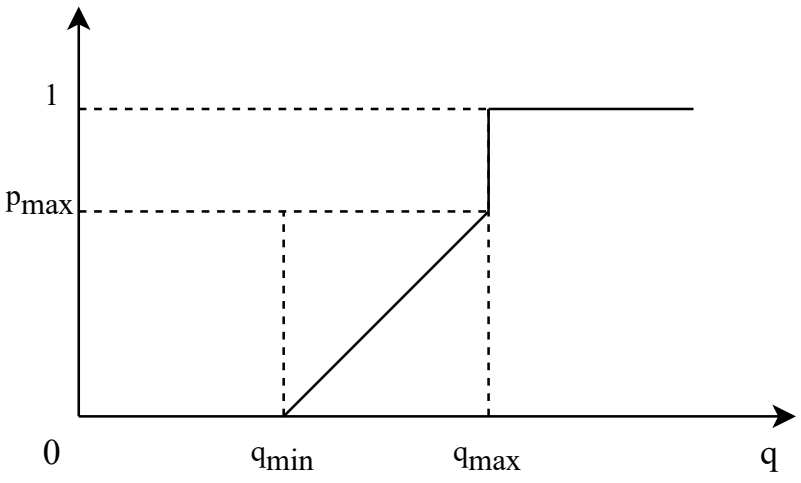

Рис. 1.1. Вид функции сброса в алгоритме RED

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 (1.1):

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\min}, \\ \frac{\hat{q} - q_{\min}}{q_{\max} - q_{\min}} p_{\max}, & q_{\min} < \hat{q} \leqslant q_{\max}, \\ 1, & \hat{q} > q_{\max}. \end{cases} \tag{1.1}$$

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.

1.2. Нелинейные и кусочно-линейные модификации

1.2.1. NLRED

Nonlinear RED — это модификация классического алгоритма RED, в котором используется нелинейная функция для определения вероятности отбрасывания пакетов.

Вероятность p_b маркировки на отбрасывание пакетов вычисляется следующим способом (1.2):

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\min}, \\ 1.5(\frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}})^2 p_{\max}, & q_{\min} < \hat{q} \leqslant q_{\max}, \\ 1, & \hat{q} > q_{\max}. \end{cases} \tag{1.2}$$

Замена линейной функции вероятности сброса пакетов на нелинейную квадратичную функцию поспособствовала тому, что, унаследовав простоту RED, NLRED менее чувствителен к настройкам параметров, имеет более предсказуемый средний размер очереди и может достичь более высокой пропускной способности. 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. [3].

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 [4].

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`.

1.2.2. GRED

GRED — алгоритм активного управления очередью, является одним из основных расширений RED. Стандартный алгоритм увеличивает вероятность отбрасывания с 0.05 до 0.5, когда средняя длина очереди увеличивается от минимального до максимального порогового значения, но при превышении максимального порога вероятность возрастает напрямую с 0.5 до 1. Этот внезапный скачок нормализуется модификацией Gentle RED, который расширяет RED тем, что добавляет дополнительное максимальное пороговое значние, которое равно $2q_{\max}$, тем самым «сглаживая» кривую [5]. 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 (1.3):

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\min}, \\ \frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}} p_{\max}, & q_{\min} \leqslant \hat{q} < q_{\max}, \\ \frac{\hat{q}-q_{\min}}{q_{\max}}(1-p_{\max}) - p_{\max}, & q_{\max} \leqslant \hat{q} < 2q_{\max}, \\ 1, & \hat{q} \geqslant q_{\max}. \end{cases} \tag{1.3}$$

1.2.3. 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 α и β соответственно, регулируемыми задаваемым селектором режимов γ [6].

Функция вероятности сброса пакетов в алгоритме DSRED показана в формуле (1.4)

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{min}, \\ \alpha\hat{q} - q_{min}, & q_{min} \leqslant \hat{q} < q_{mid}, \\ 1 - \gamma + \beta\hat{q} - q_{mid}, & q_{mid} \leqslant \hat{q} < q_{max}, \\ 1, & \hat{q} \geqslant q_{max}. \end{cases} \tag{1.4}$$

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 γ. Регулируя только γ, DSRED способен достичь высокой скорости выброса, за которой следует низкая скорость выброса, или наоборот, обеспечивая гибкую схему управления для решения сложных ситуаций с сетевой конгестией.

DSRED похож на RED в двух аспектах. Во-первых, оба алгоритма используют линейные функции сброса для обеспечения плавно увеличивающегося действия сброса на основе средней длины очереди. Во-вторых, они используют ту же функцию для расчета средней длины очереди. В результате вышеуказанных двух сходств DSRED наследует преимущества RED. Однако двухсегментная функция сброса DSRED обеспечивает гораздо более гибкую операцию сброса, чем RED. Двухсегментная функция сброса DSRED использует среднюю длину очереди, которая связана с уровнем конгестии в долгосрочной перспективе. С увеличением конгестии вероятность сброса пакетов будет увеличиваться с более высокой скоростью, а не с постоянной скоростью (как в RED). Это даст раннее предупреждение хостам о необходимости снижения нагрузки, предотвращая ухудшение конгестии.

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`.

1.2.4. 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 [7]. 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 (1.5):

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

1.2.5. 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 [8]. 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 (1.6), где $\delta = (q_{max} - q_{min})/3$.

$$p_b = \begin{cases} 0, & 0 \leqslant \hat{q} < q_{\min}, \\ 9(\frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}})^3 p_{\max}, & q_{\min} \leqslant \hat{q} < q_{\min+\delta}, \\ (\frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}}) p_{\max}, & q_{\min} + \delta \leqslant \hat{q} < q_{\min+2\delta}, \\ 9(\frac{\hat{q}-q_{\min}}{q_{\max}-q_{\min}})^3 p_{\max} + p_{\max}, & q_{\min} + 2\delta \leqslant \hat{q} < q_{\max}, \\ 1, & \hat{q} \geqslant q_{\max}. \end{cases} \tag{1.6}$$

1.2.6. 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]. Дополнительный параметр, используемый в QRED, способствует повышению его эффективности при большом количестве источников для передачи данных, уменьшая задержку, потерю пакетов при одинаковой пропускной способности. По умолчанию модификация не реализована в NS-2, для её реализации дополнил фунцию `double REDQueue::calcula` а в программе очереди указал значение переменной `quadratic_linear_ 1`..

Вероятность p_b маркировки на отбрасывание приведена в (1.7), где $Target = 2(q_{max} + q_{min})/3 - q_{min}$.

$$p_b = \begin{cases} 0, & 0 \leqslant \hat{q} < q_{\min}, \\ 9(\frac{\hat{q}-q_{min}}{2(q_{\max}-2q_{\min})})^2 p_{\max}, & q_{\min} \leqslant \hat{q} < Target, \\ p_{max} + 3(1-p_{max})(\frac{\hat{q}-Target}{q_{\max}+q_{\min}}), & Target \leqslant \hat{q} < q_{max}, \\ 1, & \hat{q} \geqslant q_{max}. \end{cases} \tag{1.7}$$

1.2.7. 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

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

Вероятность p_b маркировки на отбрасывание приведена в (1.8), где $Target = (q_{max} - q_{min})/2 + q_{min}$.

$$p_b = \begin{cases} 0, & 0 \leqslant \hat{q} < q_{\min}, \\ (\frac{\hat{q}-q_{min}}{q_{\max}-q_{\min}})^2 p_{\max}, & q_{\min} \leqslant \hat{q} < Target, \\ \sqrt{\frac{\hat{q}-q_{min}}{q_{\max}-q_{\min}}} p_{\max}, & Target \leqslant \hat{q} < q_{max}, \\ 1, & \hat{q} \geqslant q_{max}. \end{cases} \tag{1.8}$$

1.3. Адаптивные модификации

1.3.1. 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 [11; 12].

Если классический RED очень зависит от выбора параметров, то для ARED параметры зависят от условий в сети. В традиционном RED выбор подходящих значений параметров для достижения этой цели чрезвычайно сложен. При небольшой нагрузке на сеть или при высоком значении параметра $p_{\max}$ среднее значение длины очереди колеблется вокруг минимального порога. При высокой нагрузке на сеть или при низком значении параметра $p_{\max}$ среднее значение длины очереди колеблется вокруг максимального порога и часто превышает его. Для алгоритма ARED параметр $p_{\max}$ изменяется в процессе работы маршрутизатора таким образом, чтобы средняя длина очереди поддерживалась между минимальным и максимальным порогами. Данный подход снижает проблему изменчивости в задержках очереди и минимизирует количество отброшенных пакетов.

Алгоритм ARED функционирует следующим образом ((1.9)), ((1.10)). Для каждого интервала `interval` (параметр) в секундах, если $\hat{q}$ больше целевой (желаемой) $\hat{q}_t$ и $p_{\max} \leqslant 0,5$, то $p_{\max}$ увеличивается на некоторую величину α; в противном случае, если $\hat{q}$ меньше целевой $\hat{q}_t$ и $p_{\max} \geqslant 0,01$, то $p_{\max}$ уменьшается в β раз:

$$p_{\max} = \begin{cases} p_{\max} + \alpha, \ \hat{q} > \hat{q}_t, \ p_{\max} \leqslant 0,5, \\ \beta p_{\max}, \ \hat{q} < \hat{q}_t, \ p_{\max} \geqslant 0,01, \end{cases} \tag{1.9}$$

$$q_{\min} + 0,4(q_{\max} - q_{\min}) < \hat{q_t} < q_{\min} + 0,6\,(q_{\max} - q_{\min})\,. \tag{1.10}$$

Для реализации модификации в NS-2 необходимо указать в настройке очереди `set adaptive_ 1` α и β задаются командами `set alpha_` и `set beta_`. Для реализации в Mininet нужно указать в tc при настройке RED дополнительно указывается adaptive.

1.3.2. RARED

Алгоритм RARED [13] является модификаций ARED, который предлагает более активно изменять вероятность сброса $p_{\max}$, чтобы иметь возможность быстрой адаптации к изменяющейся экспоненциально взвешенной скользящей средней длине очереди $\hat{q}$. Данная модификация имеет много преимуществ над RED и ARED с точки зрения скорости потери пакетов и полезной пропускной способности.

Функции изменения параметра $p_{\max}$ представлена ниже((1.11)), ((1.12)):

$$p_{max} = \begin{cases} p_{\max} + \alpha, & \hat{q} > \hat{q_t}, \quad p_{max} \leqslant 0,5, \\ \beta p_{\max}, & \hat{q} \leqslant \hat{q_t}, \quad p_{max} > 0,5, \end{cases} \tag{1.11}$$

$$\begin{cases} q_{\min} + 0,48\,(q_{\max} - q_{\min}) < \hat{q_t} < q_{\min} + 0,52\,(q_{\max} - q_{\min})\,, \\ \alpha = \left(0,25\dfrac{\hat{q} - \hat{q_t}}{\hat{q_t}}\right) p_{\max}, \\ \beta = 1 - \left(0,17\dfrac{\hat{q} - \hat{q_t}}{\hat{q_t} - q_{\min}}\right). \end{cases} \tag{1.12}$$

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 [14].

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`.

1.3.3. POWARED

Powared является еще одной модификацией алгоритма ARED [15]. В данной модификации величина p_{max} максимального сброса считается следующим образом (1.13).

$$p_{max} = \begin{cases} p_{max} - \delta_1, & q_{\min} \leqslant \hat{q} < q_{mid}, \\ p_{max} + \delta_2, & q_{mid} < \hat{q} \leqslant q_{max}, \\ p_{max}, & \hat{q} = q_{mid}, \end{cases} \tag{1.13}$$

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

Алгоритм POWARED более агрессивно реагирует на изменение средней очереди, чем ARED. POWARED способен уменьшать как переполнение, так и недозаполнение очереди. Он превосходит RED и ARED с точки зрения использования канала и скорости потери пакетов. Хотя POWARED довольно нечувствителен к настройкам своих параметров, вопрос оптимизации его настроек остается довольно сложным.

Стратегия, принятая POWARED, заключается в том, чтобы привести средний размер очереди буфера q_{mid} к размеру, в котором система находится в точке равновесия рабочего состояния или в устойчивом состоянии. Когда средний размер очереди сходится к q_{mid}, система стабильна, и ее производительность может быть оптимизирована. POWARED требует лишь простого периодического обновления p_{max} каждые несколько секунд, что аналогично ARED, делая накладные расходы малыми и приемлемыми для внедрения в высокоскоростных маршрутизаторах, вводя только два параметра: фактор степени k и коэффициент сжатия β.

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`. Параметр модификации $k > 1$ задается с помощью переменных `pwk_`, по умолчанию $k = 2$.

1.3.4. FARED

FARED — это алгортитм, который сохраняет целевой диапазон, указанный в алгоритме RARED, но изменяет верхнюю и нижнюю границы для α и β соответственно. Алгоритм FARED обеспечивает надежную производительность в широком диапазоне сред, включая сценарии с умеренной и высокой нагрузкой на трафик [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 RARED, он может быть развернут без каких-либо сложностей (1.14), (1.15).

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`.

$$p_{max} = \begin{cases} p_{\max} + \alpha, & \hat{q} > \hat{q}_t, \quad p_{max} \leqslant 0,5, \\ \beta p_{\max}, & \hat{q} \leqslant \hat{q}_t, \quad p_{max} > 0,5, \end{cases} \tag{1.14}$$

$$\begin{cases} q_{\min} + 0,48\,(q_{\max} - q_{\min}) < \hat{q}_t < q_{\min} + 0,52\,(q_{\max} - q_{\min})\,, \\ \alpha = \left(0,0412 \dfrac{\hat{q} - \hat{q}_t}{\hat{q}_t}\right) p_{\max}, \\ \beta = 1 - \left(0,0385 \dfrac{\hat{q} - \hat{q}_t}{\hat{q}_t - q_{\min}}\right). \end{cases} \tag{1.15}$$

Глава 2. Средства моделирования сетей передачи данных

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

2.1. 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 состоит из двух частей: компонент, реализованный на языке C++, требующий перекомпиляции при модификации, и часть, написанная на языке Objective Tcl, не нуждающаяся в компиляции. Обе части имеют пересекающиеся классы, называемые в терминах NS-2 компилируемой и интерпретируемой иерархиями соответственно. Взаимодействие между данными частями регулируется спецификацией, обеспечивающей вызовы методов из одной части в другую и обратно. Компилируемая часть предназначена для обработки задач, требующих высокой производительности, в то время как интерпретируемая часть управляет моделированием и манипулированием объектами. Данный подход позволяет легко создавать модели сетей с использованием Tcl, в то время как для изменения в компилируемой части требуется модификация кода на C++ и последующей перекомпиляция. Данная архитектура обеспечивает гибкость и расширяемость средства моделирования, например возможность самостоятельной реализации алгоритма новых дисциплин управления очередью.

NS-2 включает в себя богатую библиотеку классов, которые обеспечивают моделирование широкого спектра сетевых элементов и поведений. Вот некоторые из ключевых элементов, которые можно моделировать с помощью NS-2

- Узлы сети: Виртуальные представления оконечных устройств, таких как компьютеры, мобильные устройства или серверы. Узлы могут генерировать, получать и перенаправлять данные в сети.

- Маршрутизаторы: Элементы, отвечающие за перенаправление пакетов данных между узлами в сети. Маршрутизаторы используют различные алгоритмы маршрутизации для определения оптимальных путей передачи данных.

- Каналы связи: Моделируют физические и виртуальные пути передачи данных между узлами. Каналы могут иметь различную пропускную способность, задержку и уровень ошибок, что позволяет исследовать влияние различных сетевых условий на производительность сети.

- Протоколы передачи данных: Поддержка множества стандартных и экспериментальных протоколов для моделирования поведения сети. Это включает в себя протоколы транспортного уровня, такие как TCP и UDP, протоколы маршрутизации, такие как OSPF и BGP, а также протоколы прикладного уровня.

Для создания моделей 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 [17; 18].

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.

2.2. Mininet

Mininet [19] — 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.

В архитектуре Mininet основные элементы включают в себя виртуальные хосты, которые функционируют как компьютеры с возможностью запуска собственных процессов и сетевых настроек. Для моделирования сетевых коммутаторов используются виртуальные коммутаторы, при этом часто применяется Open vSwitch для поддержки программно-конфигурируемой сетевой среды через OpenFlow. Взаимосвязь между хостами и коммутаторами осуществляется через виртуальные соединения, имитирующие физические кабельные подключения с помощью виртуальных Ethernet-интерфейсов. Контроль над сетевым трафиком обеспечивается через контроллеров, которые могут быть как встроенными, так и подключаться внешними устройствами, облегчая тестирование различных стратегий управления сетью. Такая структура позволяет Mininet обеспечивать высокую степень изоляции и контроля над сетевым окружением, используя при этом пространства имен и виртуальные интерфейсы для создания масштабируемых и гибких сетевых топологий на одном компьютере или сервере.

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.

Разберем сторонние программные средства, которые можно использовать совместно с mininet

2.2.1. Iperf3

iPerf3 [20] — это кроссплатформенное клиент-серверное приложение с открытым исходным кодом, разработанный для оценки пропускной способности сети между двумя устройствами в сети. Данный инструмент позволяет проводить тестирование с использованием различных транспортных протоколов, включая TCP, UDP и SCTP, что обеспечивает гибкость при анализе сетевых характеристик в разнообразных условиях.

Для протоколов TCP и SCTP iPerf3 предлагает следующие функции:

- Оценка пропускной способности, что позволяет измерить максимальную скорость передачи данных между двумя узлами.

- Возможность настройки размера MSS и MTU, обеспечивая тем самым способ оптимизации и адаптации сетевого взаимодействия под специфические условия.

- Мониторинг размера окна перегрузки TCP, что важно для анализа поведения TCP при управлении перегрузками в сети.

При работе с UDP, iPerf3 предоставляет следующие возможности:

- Измерение пропускной способности для определения максимальной скорости передачи данных без установления соединения.

- Оценка потерь пакетов, что критически важно для приложений, чувствительных к потерям, например, для голосовых и видео приложений.

- Измерение колебания задержки, позволяющее оценить стабильность сетевой задержки, что особенно актуально для реального времени мультимедийных приложений.

- Поддержка групповой рассылки пакетов, расширяющая возможности тестирования сети за счет эмуляции трансляций для множества получателей.

Таблица 2.1.

Основные параметры iperf3

Параметр	Значение
-c <hostname/IP>	запуск Iperf в режиме клиента, указывая адрес сервера (hostname или IP), с которым следует соединиться
-s	запуск Iperf в режиме сервера, ожидая входящих соединений от клиента
-D	запуск сервера в фоновом режиме
-p <port>	назначение порта, который будет использоваться для соединения, по умолчанию 5201
-t <seconds>	продолжительность теста в секундах, по умолчанию — 10 секунд
-J	вывод результатов теста в формат .json
-1	завершение работы сервера после передачи данных
-i <seconds>	интервал между периодическими отчётами о пропускной способности; например, -i 1 будет выводить отчёт каждую секунду

Основные парметры iperf3 представлены в таблице 2.1

Таким образом, iPerf3 представляет собой мощный инструмент для комплексного анализа производительности сети, позволяя исследователям и инженерам тонко настраивать и оценивать характеристики сетевой инфраструктуры.

2.2.2. tc

Linux Traffic Control (tc) [21] из пакета iproute2 представляет собой мощный инструмент для управления трафиком в операционных системах на базе Linux. Она позволяет контролировать поток данных через сетевые интерфейсы, обеспечивая возможность управления пропускной способностью, задержками, потерями пакетов и другими параметрами сетевого трафика. Позволяет получить данные осостоянии интерфейса

Ключевые компоненты

- qdisc: основной компонент для контроля над трафиком. Определяет, как система управляет и отправляет пакеты. Существует несколько типов qdisc, каждый из которых предоставляет различные механизмы управления очередями.
- class: используется вместе с некоторыми типами qdisc (например, HTB или Hierarchy Token Bucket) для создания иерархии и дополнительного управления пропускной способностью.
- filter: фильтры применяются для классификации трафика и его распределения между различными классами или qdisc.

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.

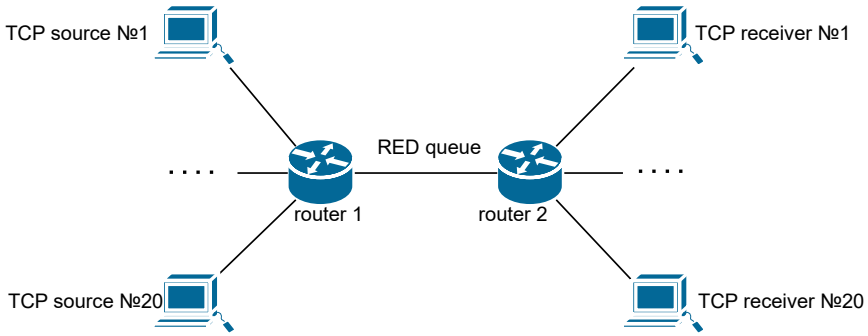

Рис. 3.1. Схема топологии моделируемой сети

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 [22].

В файле nodes.tcl создаются узлы, задаются их соединения, а также настраиваются агенты и приложения на данные узлы. В файле queue.tcl на соеденение между маршрутизаторами накладывается RED, и описываются все его метрики. В файле TCP.tcl прописана функция для мониторинга таких метрик TCP, как cwnd, rtt и rtt_var. В файле nam.tcl представлено создание nam файла для визуализации топологии. В файле timing.tcl задаются время моделирования. В файле finish.tcl описана процедура завершения. Все данные выводятся в каталог output, а графики в формате pdf с помощью скрипта GNUPLOT выводятся в каталог results. Все эти скрипты, запускающиеся с помощью головного файла main.tcl, представлены в Приложении.

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

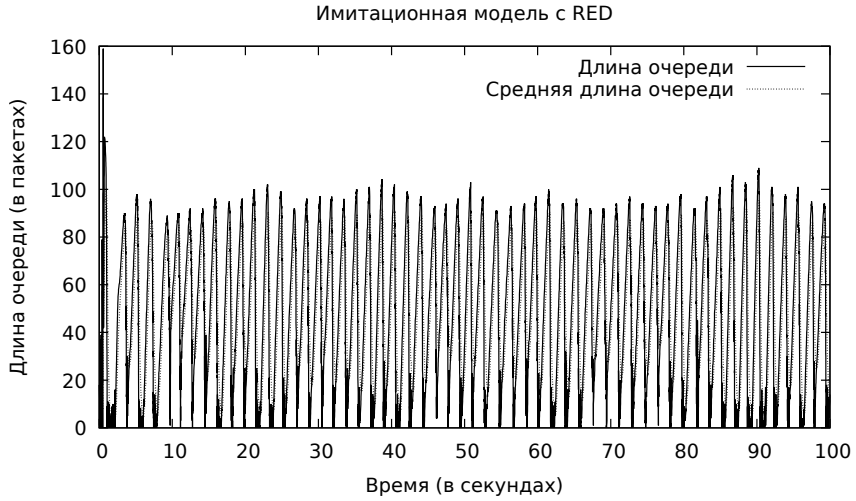

Рис. 3.2. График длины очереди и средней длины очереди на линке между маршрутизаторами

Как мы видим, в процессе моделирования очередь циклично варьируется от 0 до 100 пакетов.

Также провели модели для трех основных модификаций RED, GRED и ARED **??**.

Как мы все можем наблюдать, адаптивный алгоритм сильно отличается от двух других алгоритмах при наших значениях, в нем длина очереди всегда выше минимального порога и держится приблизительно в промежутке от 80 до 100 пакетов. Это обьяснется его кардинально отличающимся алгоритмом, меняющим параметры в зависимости от сетевой нагрузки. Монитроив также отброшенные пакеты, мы видим, что классический алгоритм

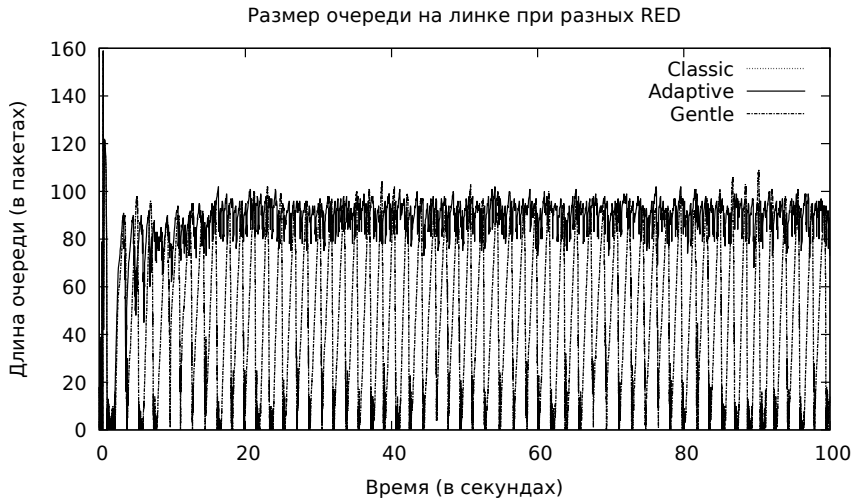

Рис. 3.3. Очередь при основных алгоритмах RED

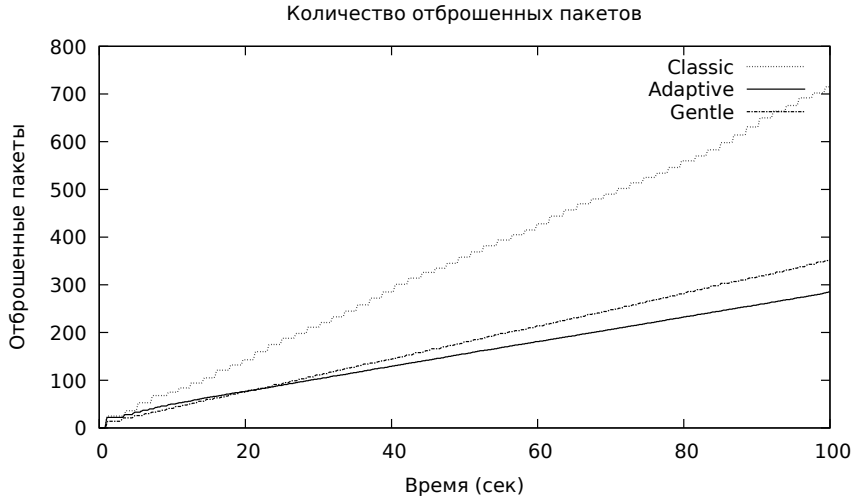

Рис. 3.4. Количество отброшенных пакетов при основных алгоритмах RED

за это время отбросил в 2 раза больше пакетов чем GRED, а ARED менее агрессивен по данному показателю, что логично так как при данной модификации длина очереди всегда выше минимального порога.

Проведя моделирования адаптивных модификаций, получили довольно похожие результаты 3.5. Для модификации POWARED параметр $k = 3$.

Как мы наблюдаем, FARED и RARED доволно похожи и мы получаем похожие значения, FARED в начале модели имеет больший диапазон очереди, но максимальный размеры у них не отличаются, так как модификации отличаются только небольшим отличием числовых параметров. А модификация POWARED при $k = 3$ сохраняет больший размер очереди и вообще не опускается ниже минимального порога.

Для проведения сравнительного анализа различных нелинейных и кусочно-линейных алгоритмов, относящихся к семейству RED, было выполнено моделирование аналогичной

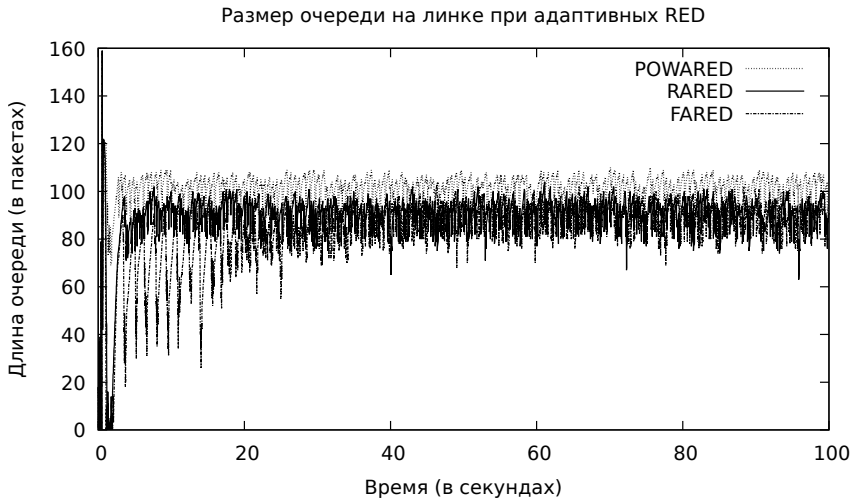

Рис. 3.5. Очередь при адаптивных алгоритмах RED

топологии, время масштабировано до первых 25 секунд для большей наглядности. В результате получили следующие значения средневзвешенной экспоненциальной очереди, см. рис. 3.6. В данном примере все алгоритмы, кроме DSRED, показали крайне схожие результаты обработки очереди. DSRED начинает отбрасывать пакеты более агрессивно из-за своего промежуточного значения q_{mid}.

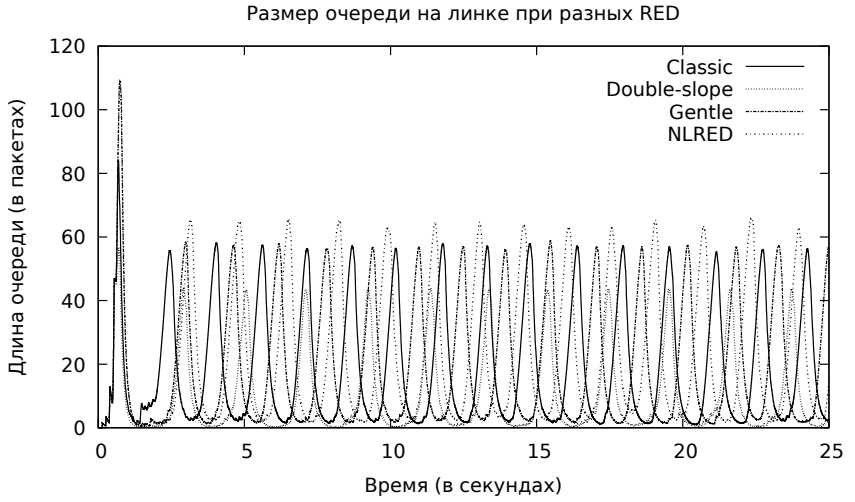

Рис. 3.6. Длина очереди при разных алгоритмах RED

3.2. Моделирование сети в Mininet

Модель в Mininet более сложная в реализации. В нем используются реальные виртуальные маршрутизаторы из класса LinuxRouter, описанных в [23] 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 [24]. Также для моделирования программа обращаяется к ядру операционной системы, для того, чтобы изменить тип и размер окна TCP. Для запуска на нескольких устройствах используются паралельные потоки. Для визуализации также используется GNUPLOT. Также стоит учитывать, что при реальном моделировании, в отличие от имитационной, результаты могут отличаться при разных запусках, в зависимости от многих факторов, как оперативная память компьютера, количество ядер процессора и т. д.

В файле topology.py описаны создание топологии, двух маршрутизаторов, двух коммутаторов и соединеных с ними динамического количества оконечных устройств. В файле router.py описан класс LinuxRouter, используемых для данной сети. В файле utils.py описаны функции запуска iperf и мониторинга интерфейса с RED. В файлах all.py и queue.py описаны функции для получения данных о метриках TCP и интерфейса с RED. В файле main.py задана ip-адресация и созданы потоки для запуска модели.

3.3. Сравнение результатов в двух средствах моделирования

Рассмотрим количество отброшенных пакетов в двух средствах. Как показано на Рисунках 3.7 и 3.8, поведение системы в NS-2 и Mininet имеет значительные различия в отношении количества отброшенных пакетов.

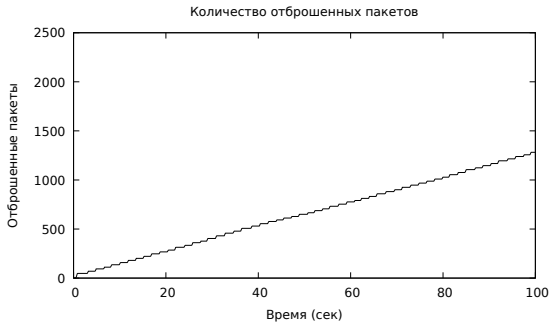

Рис. 3.7. Отброшенные пакеты в NS-2

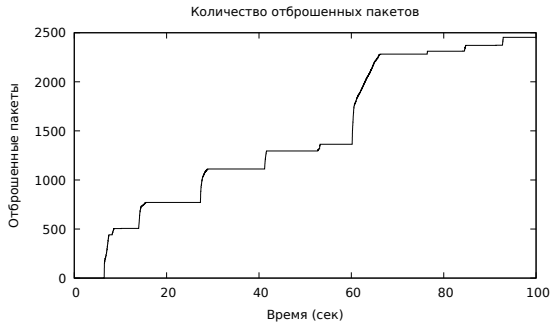

Рис. 3.8. Отброшенные пакеты в mininet

Как мы видим, при натурном моделировании отбрасываются больше пакетов, а более плавное изменение графика в NS-2 обьяснется тем, что мониторинг данных происходит с большей частотой.

Заключение

В ходе выполнения выпускной квалификационной работы был дан обзор основных алгоритмов семейства RED. Был рассмотрены средства моделирования NS-2 и Mininet и изучены существующие в них инструменты для моделирования сетей передачи данных.

Разработан программный комплекс, реализующий имитационной и натурной моделей заданной топологии [25]. Для визуализации результатов, а также для оценки работы было использована программа GNUPLOT. Были описанны структура и все функции, из которых состоит программный комплекс.

Таким образом были выполнены задачи, сформулированные перед началом работы:

1. Разобрано архитектуры средств моделирования и составлены алгоритмы работы в данных симуляторах.

2. Разработан программный комплекс, позволяющий моделировать разные топологии сети с дисциплиной RED.

3. Проведён анализ результатов моделирования сети с управлением очередями на маршрутизаторе и сделан вывод об эффективности работы алгоритма RED при запуске в разных средах.

Список литературы

1. *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.
2. *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.
3. *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.
4. *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.
5. 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.
6. *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.
7. *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.
8. 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.
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. 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.
11. *Floyd S.*, *Gummadi R.*, *Shenker S.* Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. — 2001. — Сент.

12. *Domańska J.*, *Domański A.* Adaptive RED in AQM // Communications in Computer and Information Science. — Springer Berlin Heidelberg, 2009. — С. 174—183. — DOI: 10.1007/978-3-642-02671-3_21.

13. *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.

14. *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.

15. POWARED for Non-Linear Adaptive RED / B. Ng [и др.] // 2005 Asia-Pacific Conference on Communications. — IEEE, 2005. — DOI: 10.1109/apcc.2005.1554179.

16. *Tahiliani M. P.*, *Shet K. C.*, *Basavaraju T. G.* FARED: Fast Adapting RED Gateways for TCP/IP Networks // Advanced Computing, Networking and Security / под ред. P. S. Thilagam [и др.]. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. — С. 435—443. — DOI: 10.1007/978-3-642-29280-4_51.

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. *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.

19. *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.

20. iPerf 3 user documentation / J. Dugan [и др.]. — 2024. — URL: https://iperf.fr/iperf-doc.php.

21. *Kerrisk M.* tc(8) — Linux manual page. — 2001. — URL: https://man7.org/linux/man-pages/man8/tc.8.html.

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

23. *Mininet*. Mininet: Rapid Prototyping for Software Defined Networks. — 2023. — URL: https://github.com/mininet/mininet/tree/master.

24. *Kfoury E.* Shell script that plots Iperf3's JSON file. — 2021. — URL: https://github.com/ekfoury/iperf3_plotter.

25. *Саргсян А. Г.* Имитационная и Натурная модели одинаковой топологии для научно-исследовательской работы. — 2024. — URL: https://github.com/agsargsyan/study_2023-2024_programs.

Приложение А. Реализация модификаций RED в ns-2

В данном приложении представлены функции на языке C++, реализующие работу алгоритмов модификаций RED в NS-2. Все наименования самостоятельно прописанных функций даны аналогично уже существующим.

А.1. Обновление вероятностной 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 = v_c * v_ave + v_d;
        } else if (!gentle && v_ave >= th_max) {
                p = 1.0;
        } else if (edp_.quadratic_linear == 1) {   //алгоритм работы
 ↪  REDQL
          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) {  //алгоритм работы IRED
          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)/(th_max + th_min));
          } else if (v_ave >= target) {
            p = max_p + 3*(1-max_p)*(v_ave-target)/(2*(th_max - 2 *
 ↪  th_min));
```

```
26            }
27          } else if (edp_.smart == 1) {      //алгоритм работы SmRED
28            target = ((th_max - th_min)/2) + th_min;
29            if(v_ave < target){
30              p = max_p * pow(((v_ave-th_min)/(th_max - th_min)), 2);
31            } else if (v_ave >= target) {
32              p = max_p * pow(((v_ave-th_min)/(th_max - th_min)), 0.5);
33            }
34          } else if (edp_.three_sections == 1){  //алгоритм работы TRED
35            double delta = (th_min+th_max/3);
36            if (v_ave < (th_min + delta)){
37              p = 9 * max_p * pow((v_ave-th_min)/(th_max-th_min), 3) ;
38              }
39            else if ((v_ave >= th_min + delta) && (v_ave < th_min + 2 *
    ↪ delta)){
40              p = max_p * (v_ave-th_min)/(th_max-th_min);
41              }
42            else if (v_ave >= th_min + 2* delta){
43              p =  9 * max_p * pow((v_ave-th_min)/(th_max-th_min), 3) +
    ↪ max_p;
44            }
45          }
46          else if (edp_.double_slope == 1) {   //алгоритм работы DSRED
47            double a = (2-2* edp_.omega)/(th_max - th_min);
48            double b = (2 * edp_.omega)/(th_max - th_min);
49            target = ((th_max + th_min)/2);
50            if(v_ave < target){
51              p = a * (v_ave-th_min);
52            } else if (v_ave >= target) {
53              p = 1 - edp_.omega + b * (v_ave - target);
54            }
55          }
56          else {
57                  p = v_a * v_ave + v_b;
58                  // p = (v_ave - th_min) / (th_max - th_min)
59
```

```
60                  /* Added by Mohit P. Tahiliani for Nonlinear RED
    ↪  (NLRED) - Start */
61          if(edp_.nonlinear == 1){
62            p *= p;     // для NLRED
63          }
64          else if (edp_.hyperbola == 1){
65            p *= 1/p;  // для HRED
66          }
67          else if (edp_.exponential == 1){ // алгоритм работы RED_e
68            p = (pow(exponenta, v_ave)-pow(exponenta,
    ↪  th_min))/(pow(exponenta, th_max)-pow(exponenta, th_min));
69          }
70                  p *= max_p;
71          }
72    if (p > 1.0)
73      p = 1.0;
74    return p;
75  }
```

A.2. Функции для адаптивных RED

```
1   void REDQueue::updateMaxP_refined_adaptive(double new_ave, double now)
2   {
3       double part = 0.48*(edp_.th_max - edp_.th_min);
4     if ( new_ave < edp_.th_min + part && edv_.cur_max_p > edp_.bottom) {
5       edv_.cur_max_p = edv_.cur_max_p * (1.0 - (0.17 * ((edp_.th_min +
    ↪  part) - new_ave) / ((edp_.th_min + part) - edp_.th_min)));
6       edv_.lastset = now;
7       double maxp = edv_.cur_max_p;
8     } else if (new_ave > edp_.th_max - part && edp_.top > edv_.cur_max_p
    ↪  ) {
9       double alpha = edp_.alpha;
10      alpha = 0.25 * edv_.cur_max_p * ((new_ave - (edp_.th_max - part))
    ↪  / (edp_.th_max - part));
11      edv_.cur_max_p = edv_.cur_max_p + alpha;
12      edv_.lastset = now;
13      double maxp = edv_.cur_max_p;
```

```
14      }
15  }
16
17  void REDQueue::updateMaxP_powared(double new_ave, double now)
18  {
19      double target = 0.5*(edp_.th_max + edp_.th_min);
20      int k = edp_.pwk;
21      int b = edp_.pwb;
22      int r = edp_.bf_size;
23      double v_ave = edv_.v_ave;
24      double delta1 = abs(pow((v_ave - target)/(b * target), k));
25      double delta2 = abs(pow((target - v_ave)/(b * (r -target)), k));
26
27     if ( new_ave < target && edv_.cur_max_p > edp_.bottom) {
28       edv_.cur_max_p = edv_.cur_max_p - delta1;
29       edv_.lastset = now;
30       double maxp = edv_.cur_max_p;
31     } else if (new_ave > target && edp_.top > edv_.cur_max_p ) {
32       edv_.cur_max_p = edv_.cur_max_p + delta2;
33       edv_.lastset = now;
34       double maxp = edv_.cur_max_p;
35     }
36  }
37
38  void REDQueue::updateMaxP_fast_adaptive(double new_ave, double now){
39        double part = 0.48*(edp_.th_max - edp_.th_min);
40      if ( new_ave < edp_.th_min + part && edv_.cur_max_p > edp_.bottom)
    ↪  {
41       edv_.cur_max_p = edv_.cur_max_p * (1.0 - (0.0385 * ((edp_.th_min +
    ↪  part) - new_ave) / ((edp_.th_min + part) - edp_.th_min)));
42       edv_.lastset = now;
43       double maxp = edv_.cur_max_p;
44     } else if (new_ave > edp_.th_max - part && edp_.top > edv_.cur_max_p
    ↪   ) {
45       double alpha = edp_.alpha;
46       alpha = 0.0412 * edv_.cur_max_p * (new_ave - part) / part;
47       edv_.cur_max_p = edv_.cur_max_p + alpha;
```

```
48        edv_.lastset = now;
49        double maxp = edv_.cur_max_p;
50      }
51  }
```

Приложение В. Имитационная модель

В данном приложении представлен исходный код для проведения имитационного моделирования передачи данных в программе NS-2 с использованием языка Objective TCL.

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 очереди)
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 $N + $t]) $t]
  $tcp($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
  ↪  исходного кода программы

#$redq set nonlinear_ 1
#$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 мгновеной 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";
        }
    }
```

```
18      set f [open output/temp.queue w]
19      puts $f "TitleText: RED"
20      puts $f "Device: Postscript"
21
22      if { [info exists tchan_] } {
23         close $tchan_
24      }
25      exec rm -f output/temp.q output/temp.a
26      exec touch output/temp.a output/temp.q
27      exec awk $awkCode output/all.q
28
29      puts $f \"queue
30      exec cat output/temp.q >@ $f
31      puts $f \n\"ave_queue
32      exec cat output/temp.a >@ $f
33      close $f
34      # вывод графиков в xgraph для быстрого просмотра
35      exec xgraph -bb -tk -x time -t "TCPRenoCWND" output/WvsT &
36      exec xgraph -bb -tk -x time -t "TCPRenoCWND_1" output/WvsT_1 &
37      exec xgraph -bb -tk -x time -t "RTT" output/RTT &
38      exec xgraph -bb -tk -x time -t "RTTVAR" output/RTTVAR &
39      exec xgraph -bb -tk -x time -y queue output/temp.queue &
40      #exec nam output/out.nam &
41      exit 0
42  }
```

Приложение С. Натурная модель

В данном приложении представлен исходный код программы для решения стохастических дифференциальных уравнений, написанный на языке С с использоваеним библиотеки GSL.

topology.py

```
from mininet.topo import Topo
from router import LinuxRouter

# создание топологии для сети
class NetworkTopo(Topo):
    def build(self, num_hosts_subnets, **_opts):
        # создание маршрутизаторы
        r0 = self.addHost('r0', cls=LinuxRouter, ip='10.0.0.1/24')
        r1 = self.addHost('r1', cls=LinuxRouter, ip='10.1.0.1/24')

        # добавление коммутаторов для связи хостов
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')

        # соединение между маршрутизаторами и коммутаторами
        self.addLink(s1, r0, intfName2='r0-eth1', params2={'ip':
         ↪  '10.0.0.1/24'}, max_queue_size=300)
        self.addLink(s2, r1, intfName2='r1-eth1', params2={'ip':
         ↪  '10.1.0.1/24'}, max_queue_size=300)

        # соединение между маршрутизаторами
        self.addLink(r0, r1, intfName1='r0-eth2', intfName2='r1-eth2',
         ↪  max_queue_size=30000, params1={'ip': '10.100.0.1/24', 'bw':
         ↪  '20'}, params2={'ip': '10.100.0.2/24', 'bw': '15'})

        # добавление оконечных устройств
        for i in range(1, 2*num_hosts_subnets + 1):
            host_ip = '10.0.0.{}/24' if i % 2 == 1 else '10.1.0.{}/24'
            default_route = 'via 10.0.0.1' if i % 2 == 1 else 'via
             ↪  10.1.0.1'
```

```
            host = self.addHost(name='h{}'.format(i),
             ↪  ip=host_ip.format(250 - i), defaultRoute=default_route)
            switch = s1 if i % 2 == 1 else s2
            self.addLink(host, switch, bw='100m')
```

utils.py

```
from threading import Thread
import threading
import time

iperf_time = 101
# Функция для запуска iperf между парой хостов
def run_iperf(net, host1_name, host2_name):

    h1 = net.get(host1_name)
    h2 = net.get(host2_name)
    h2.cmd("iperf3 -s -D")
    #time.sleep(5)
    h1.cmd(f"mkdir -p output/{host1_name}_to_{host2_name}")
    h1.cmd(f'iperf3 -c {h2.IP()} -w 40000 -t {iperf_time} -l 1000 -J >
     ↪  output/{host1_name}_to_{host2_name}/{host1_name}_to_{host2_name}_iperf
    h1.cmd(f"cd output/{host1_name}_to_{host2_name} && plot_iperf.sh
     ↪  {host1_name}_to_{host2_name}_iperf3.json")

# Функция для запуска мониторинга на линке с RED
def monitor_queue(net, interface="r0-eth2", interval=0.025,
 ↪  output_file='queue_monitor.txt'):
    t = threading.current_thread()
    with open(output_file, 'w') as file:
        while getattr(t, "do_run", True):
            result = net['r0'].cmd(f'tc -s qdisc show dev {interface}')
            print(result, file=file)
            #time.sleep(interval)

```

Список иллюстраций

Список таблиц