

**Федеральное государственное автономное образовательное
учреждение высшего образования
«Российский университет дружбы народов имени Патриса
Лумумбы»**

Факультет физико-математических и естественных наук
Кафедра прикладной информатики и теории вероятностей

Направление: 09.03.03 Прикладная информатика

ОТЧЕТ

о прохождении учебной практики
Научно-исследовательская работа

(получение первичных навыков научно-исследовательской работы)

Место прохождения практики: отдел технической поддержки пользователей
(департамент технологических и информационных ресурсов) РУДН и
научные центры института прикладной математики и телекоммуникаций

ФИО: Саргсян Арам Грачьевич
Курс, группа 4, НПИбд-02-20
Студенческий билет № 1032201740

Руководители практики:
от РУДН к.ф.-м.н. Е.Г. Медведева

Научный руководитель:
к.ф.-м.н., доц. А.В. Королькова

Руководитель от организации:
д.т.н., проф. К.Е. Самуйлов

Оценка _____

г. Москва
2023 г.

Оглавление

Введение	3
1 Методы и материалы	5
1.1 NS-2	5
1.2 Mininet	5
1.3 Cisco Packet Tracer	6
1.4 GNS-3	6
2 RANDOM EARLY DETECTION	8
2.1 RED	8
2.2 GRED	9
2.3 WRED	11
2.4 NLRED	11
2.5 HRED	12
2.6 TRED	13
2.7 RED-QL	14
2.8 SmRED	14
2.9 DS-RED	15
2.10 ARED	16
2.11 RARED	16
2.12 Powared	17
2.13 FARED	18
3 Результаты	19
Заключение	24
Список литературы	26
Приложения	28

Введение

Согласно программе учебной практики по направлению 09.03.03 «Прикладная информатика» целями практики являются:

- формирование навыков использования современных научных методов для решения научных и практических задач;
- формирование универсальных, общепрофессиональных и профессиональных компетенций в соответствии с ОС ВО РУДН;
- формирование навыков проведения исследовательской работы;
- формирование навыков работы с источниками данных;
- знакомство с принципами функционирования и изучение методов разработки и анализа моделей функционирования сложных систем, их фрагментов и отдельных элементов;
- применение методов для анализа и расчёта показателей функционирования сложных систем, их фрагментов и отдельных элементов.

Также определены задачи практики:

- изучение специфики функционирования и соответствующих методов анализа сложных систем;
- формирование навыков решения конкретных научно-практических задач самостоятельно или в научном коллективе;
- формирование навыков проведения исследовательской работы и получении научных и прикладных результатов;
- изучение принципов и методов построения моделей сложных систем (в том числе технических систем, сетей и систем телекоммуникаций);

- изучение принципов и методов анализа поведения параметров моделей сложных систем (в том числе программных и технических систем, сетей и систем телекоммуникаций, и т.п.);
- приобретение практических навыков в области изучения научной литературы и (или) научно-исследовательских проектов в соответствии с будущим профилем профессиональной области.

Для достижения вышеупомянутых целей и задач в рамках учебной практики по теме «Моделирования алгоритма управления очередями RED в среде моделирования NS-2» мною было выполнено следующее:

- рассмотрены основные методы имитационного, аналитического и натурального моделирования сетей;
- исследована специфика моделирования различных сетей с помощью программы NS-2;
- проведен сравнительный анализ результатов имитационного моделирования сети (построены и проанализированы графики размера TCP-окна, длины очереди и средней взвешенной длины очереди) при различных модификациях алгоритма RED, разных пороговых значений и типов TCP.

1 Методы и материалы

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

1.1 NS-2

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

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

1.2 Mininet

Mininet — это симулятор сетевых топологий на основе виртуализации, 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.

1.3 Cisco Packet Tracer

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

1.4 GNS-3

GNS-3 — это программное средство моделирования сетей, позволяющий создавать виртуальные сети, состоящие из реальных или виртуальных устройств, и анализировать их поведение. GNS-3 разработан на языке программирования Python и основан на эмуляторе динамических узлов Dynamips, который позволяет запускать реальные образы операционных систем. В отличие от Packet Tracer, GNS-3 позволяет смоделировать не только устройства Cisco, но и другие устройства, например, Juniper, Palo, Alto и другие, что позволяет смоделировать различные типы сетей, включая центры обработки данных и облачные инфраструктуры. Одной из главных особенностей GNS-3 является интеграция с виртуальными машинами, что расширяет возможности моделирования. Появляется возможность создавать сетевые сценарии,

в которых виртуальные машины выполняют реальные функции, такие как серверы, клиенты, точки доступа Wi-Fi и т.д. Это позволяет проводить натурное моделирование и получить более реалистичные результаты в рамках виртуальной среды.

2 RANDOM EARLY DETECTION

2.1 RED

Random Early Detection (RED) — это механизм предотвращения перегрузки на шлюзе. Он основан на общих принципах, полезен для управления средним размером очереди в сети, где не доверяют взаимодействию между протоколами передачи данных. В отличие от Droptail, который работает таким образом, что когда очередь заполняется, новые пакеты, поступающие в очередь, начинают теряться, алгоритм RED учитывает потоки трафика в сети и стремится предоставить равную пропускную способность для каждого соединения, что позволяет избежать перегрузки сети и улучшить качество обслуживания. В оригинальном RED маршрутизатор вычисляет усредненный по времени средний размер очереди с использованием фильтра нижних частот (экспоненциально взвешенное скользящее среднее) или сглаживания по длине выборки очередей, средний размер очереди сравнивается с двумя пороговыми значениями: минимальным порогом и максимальным. Когда средний размер очереди меньше минимального порога, пакеты не отбрасываются, когда средний размер очереди превышает максимальный порог, отбрасывается все поступающие пакеты. Если размер средней очереди находится между минимальным и максимальным порогом, пакеты отбрасываются с вероятностью p , которая линейно увеличивается до тех пор, пока средняя очередь не достигнет максимального порога. Подробно алгоритм описан в [1, 2].

Вероятность p 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):

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

График функции вероятности потери пакета в зависимости от среднего размера очереди представлен на рис. 2.1.

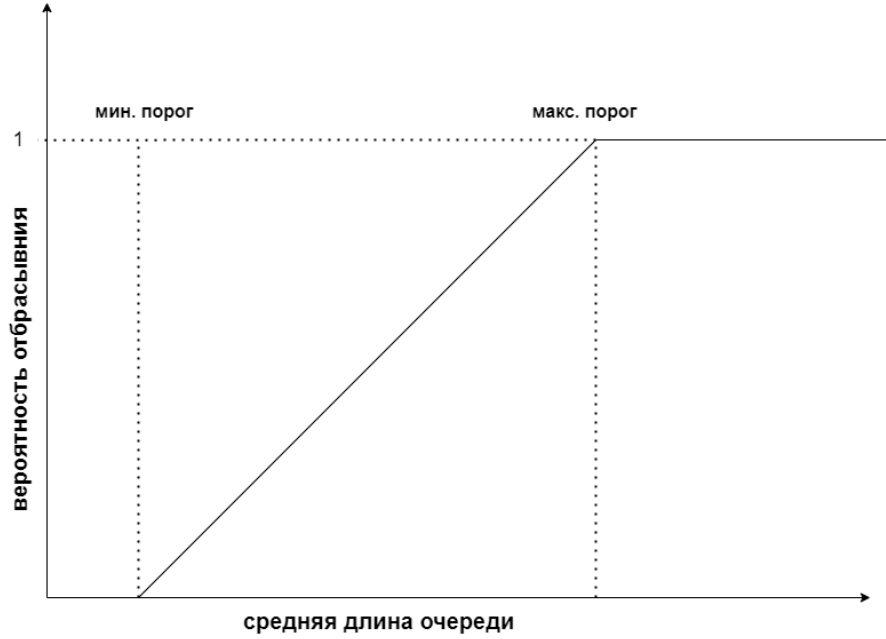


Рис. 2.1: Классический RED

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

2.2 GRED

GRED (Gentle Random Early Detection, мягкое/аккуратное произвольное раннее обнаружение) — алгоритм активного управления очередью, является рас-

ширением 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 значение, которое равно $2q_{\max}$, тем самым «сглаживая» кривую [3]. 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 является основной, так как переменная `gentle_` по умолчанию является истинной.

Вероятность сброса определяется следующим образом (2.2):

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

График функции вероятности потери пакета в зависимости от среднего размера очереди выглядит следующим образом (рис. 2.2):

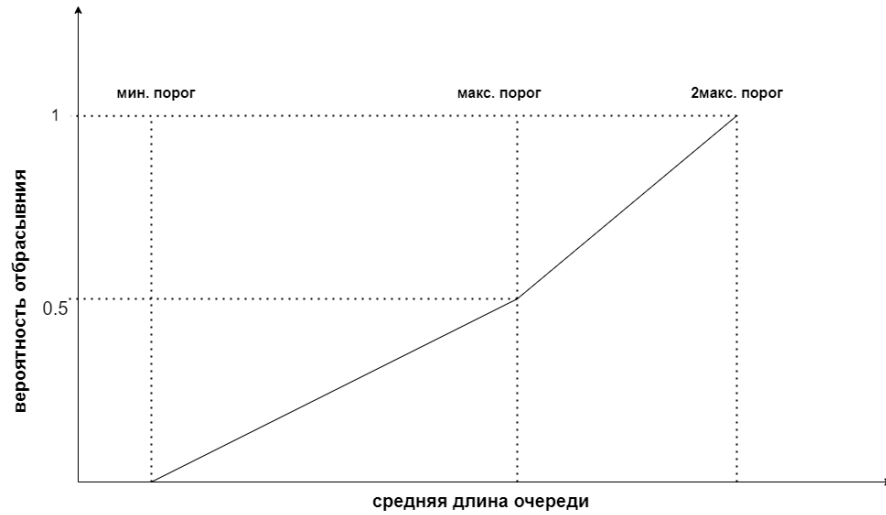


Рис. 2.2: Gentle RED

2.3 WRED

WRED (Weighted random early detection — взвешенное произвольное раннее обнаружение) — это алгоритм активного управления очередью, является расширением RED [4].

Взвешенный алгоритм произвольного раннего обнаружения предоставляет различные уровни обслуживания пакетов в зависимости от вероятности их отбрасывания и обеспечивает избирательную установку параметров механизма RED на основании типа трафика (рис. 2.3).

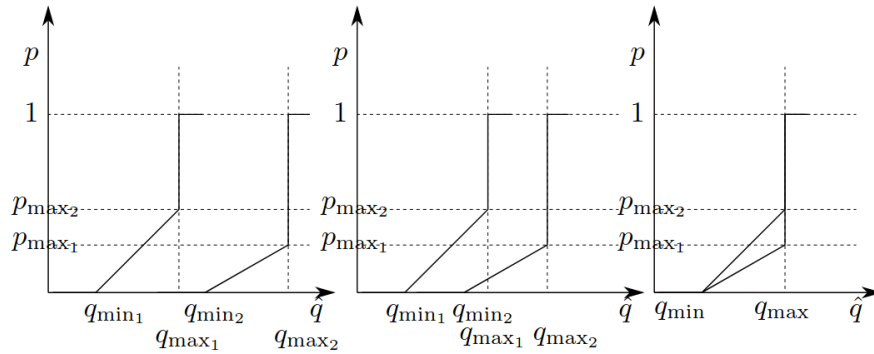


Рис. 2.3: Weighted RED

2.4 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. [5, 6].

Вероятность p_b маркировки на отбрасывание пакетов вычисляется следующим способом (2.4):

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leq q_{\min}, \\ 1.5 \left(\frac{\hat{q} - q_{\min}}{q_{\max} - q_{\min}} \right)^2 p_{\max}, & q_{\min} < \hat{q} \leq q_{\max}, \\ 1, & \hat{q} > q_{\max}. \end{cases} \quad (2.3)$$

График функции вероятности потери пакета в зависимости от среднего размера очереди представлен на рис. 2.4.

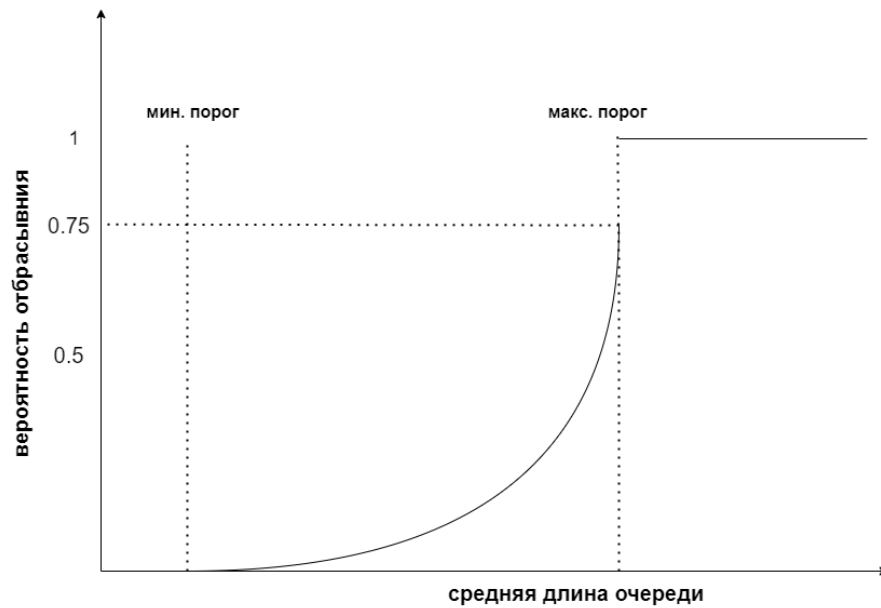


Рис. 2.4: NonLinear RED

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

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

2.5 HRED

HRED(Hyperbola random early detection) —это модификация классического RED с нилейно возрастающей функцией отбрасывания пакетов в сети. HRED менее чувствителен к настройкам параметров, чем другие схемы. При заданном значении q_{\max} HRED ведет себя похожим образом и не сильно зависит

от других параметров. HRED может достичь более высокого использования сети. HRED обеспечивает предсказуемую задержку в очереди сети. HRED сохраняет способность контролировать кратковременную перегрузку путем поглощения пакетных потоков, так как он все еще использует алгоритм подсчета среднего размера очереди и поддерживает неполную очередь. Размер очереди может быть задан и зависит от требований. HRED прост в реализации и легко внедряется на маршрутизаторах, так как меняется только профиль отбрасывания по сравнению с классическим алгоритмом RED [7]. 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 `hyperbola_1`.

Вероятность p_b маркировки на отбрасывание пакетов вычисляется следующим способом (2.4):

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

2.6 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. [8]. По умолчанию модификация не реализована в NS-2, для её реали-

зации дополнил функцию `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 (2.5), где $\delta = (q_{max} - q_{min})/3$.

$$p_b = \begin{cases} 0, & 0 \leq \hat{q} < q_{min}, \\ 9\left(\frac{\hat{q}-q_{min}}{q_{max}-q_{min}}\right)^3 p_{max}, & q_{min} \leq \hat{q} < q_{min} + \delta, \\ \left(\frac{\hat{q}-q_{min}}{q_{max}-q_{min}}\right) p_{max}, & q_{min} + \delta \leq \hat{q} < q_{min} + 2\delta, \\ 9\left(\frac{\hat{q}-q_{min}}{q_{max}-q_{min}}\right)^3 p_{max} + p_{max}, & q_{min} + 2\delta \leq \hat{q} < q_{max}, \\ 1, & \hat{q} \geq q_{max}. \end{cases} \quad (2.5)$$

2.7 RED-QL

RED-QL(Random early detection-quadratic linear) — модификация алгоритма RED, также является разновидностью алгоритма с нелинейно возрастающей функцией. RED-QL имеет квадратично-линейную форму и определяется на основе параметров, которые могут быть настроены для определенных требований сети[9] 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 `quadratic_linear_ 1`.

Вероятность p_b маркировки на отбрасывание приведена в (2.6), где $Target = 2(q_{max} + q_{min})/3 - q_{min}$.

$$p_b = \begin{cases} 0, & 0 \leq \hat{q} < q_{min}, \\ 9\left(\frac{\hat{q}-q_{min}}{2(q_{max}-2q_{min})}\right)^2 p_{max}, & q_{min} \leq \hat{q} < Target, \\ p_{max} + 3(1 - p_{max})\left(\frac{\hat{q}-Target}{q_{max}+q_{min}}\right), & Target \leq \hat{q} < q_{max}, \\ 1, & \hat{q} \geq q_{max}. \end{cases} \quad (2.6)$$

2.8 SmRED

SmRED(Smart random early detection) — модификация RED, в которой вероятность отбрасывания пакетов регулируется в зависимости от нагрузки трафика для достижения оптимальной сквозной производительности. Кроме того, переход с RED на SmRED в реальной сети требует очень мало работы из-за

своей простоты. SmRED эффективно устраняет недостатки RED, увеличивает пропускную способность при низкой нагрузке и уменьшает задержку при высокой нагрузке [10]. 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 (2.7), где $Target = (q_{max} - q_{min})/2 + q_{min}$.

$$p_b = \begin{cases} 0, & 0 \leq \hat{q} < q_{min}, \\ \left(\frac{\hat{q} - q_{min}}{q_{max} - q_{min}}\right)^2 p_{max}, & q_{min} \leq \hat{q} < Target, \\ \sqrt{\frac{\hat{q} - q_{min}}{q_{max} - q_{min}}} p_{max}, & Target \leq \hat{q} < q_{max}, \\ 1, & \hat{q} \geq q_{max}. \end{cases} \quad (2.7)$$

2.9 DS-RED

Алгоритм DS-RED (Double slope random early detection) — это ещё одна модификация RED, в котором вводится дополнительное пороговое значение q_{mid} между минимальным q_{min} и максимальным RED. Функция сброса описывается двумя линейными сегментами с углами наклона α и β соответственно, регулируемые задаваемым селектором режимов γ [11]. 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`.

Функция вероятности сброса пакетов в алгоритме DSRED показана в (2.8)

$$p_b = \begin{cases} 0, & 0 < \hat{q} \leq q_{min}, \\ \alpha \hat{q} - q_{min}, & q_{min} \leq \hat{q} < q_{mid}, \\ 1 - \gamma + \beta \hat{q} - q_{mid}, & q_{mid} \leq \hat{q} < q_{max}, \\ 1, & \hat{q} \geq q_{max}. \end{cases} \quad (2.8)$$

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

2.10 ARED

В алгоритме Adaptive RED (ARED) функция сброса модифицируется посредством изменения по принципу AIMD, заключающейся в том, что увеличение некоторой величины производится путём сложения с некоторым параметром, а уменьшение — путём умножения на параметр [12]. Для её реализации в NS-2 необходимо указать в настройке очереди `set adaptive_ 1,`.

Алгоритм ARED функционирует следующим образом (2.9), (2.10). Для каждого интервала `interval` (параметр) в секундах, если \hat{q} больше целевой (желаемой) \hat{q}_t и $p_{\max} \leq 0,5$, то p_{\max} увеличивается на некоторую величину α ; в противном случае, если \hat{q} меньше целевой \hat{q}_t и $p_{\max} \geq 0,01$, то p_{\max} уменьшается в β раз, α и β задаются командами `set alpha_` и `set beta_`:

$$p_{\max} = \begin{cases} p_{\max} + \alpha, & \hat{q} > \hat{q}_t, \ p_{\max} \leq 0,5, \\ \beta p_{\max}, & \hat{q} < \hat{q}_t, \ p_{\max} \geq 0,01, \end{cases} \quad (2.9)$$

$$q_{\min} + 0,4(q_{\max} - q_{\min}) < \hat{q}_t < q_{\min} + 0,6(q_{\max} - q_{\min}). \quad (2.10)$$

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 $\beta > 0.83$.

2.11 RARED

Алгоритм Refined ARED является модификацией ARED и предлагает более активно изменять вероятность сброса p_{\max} , чтобы иметь возможность

быстрой адаптации к изменяющейся экспоненциально взвешенной скользящей средней длине очереди \hat{q} [13].

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

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

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

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

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

2.12 Powared

Powared является модификацией алгоритма ARED [14]. 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::updateM` а в настройке очереди сети указал значение переменных `adaptive_ 1` и `powared_ 1`. Параметры модификации задаются с помощью переменных `pwk_`, `pwb_`.

$$p_{max} = \begin{cases} p_{max} - \delta_1, & q_{min} \leq \hat{q} < q_{mid}, \\ p_{max} + \delta_2, & q_{mid} < \hat{q} \leq q_{max}, \\ p_{max}, & \hat{q} = q_{mid}, \end{cases} \quad (2.13)$$

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

2.13 FARED

FARED(Fast Adapting RED) — это алгоритм, 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 [15]. 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`.

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

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

3 Результаты

Мною был написана программа, реализующая имитационную модель сети со следующей топологией:

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

Полная реализация программы приведена в разделе **Приложения**, для вывода графиков была использована программа GNUPLOT.

Смоделировав сеть с указанными параметрами и запустив gnuplot-скрипт, я получил следующий график (рис. 3.1).

Как мы видим из первого графика, в момент времени $t = 1$ с достигается максимальные длины очереди 140000 пакетов и средней длины очереди

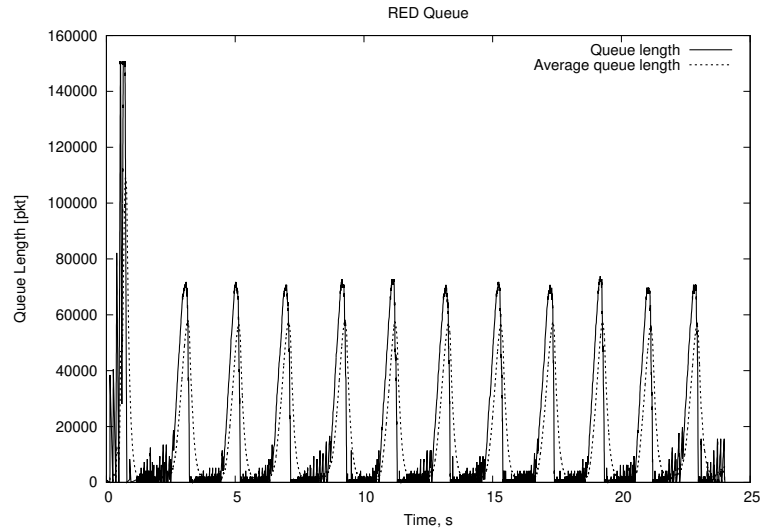


Рис. 3.1: График длины очереди и средней длины очереди на линке (R0–R1) ($q_{\min} = 75$, $q_{\max} = 150$, $q_w = 0.002$, $p_{\max} = 0.1$, TCP типа TCP/Reno)

120000 пакетов, а при дальнейшем моделировании длина очереди варьируются от 0 до 70000 пакетов, а средняя очередь от 0 до 60000, наступает стационарное состояние.

Для выявления влияния пороговых значений на длину очереди смоделировал сеть и вывел на одном графике траектории средней длины очереди при разных пороговых значениях и при одинаковых остальных параметрах (рис. 3.2).

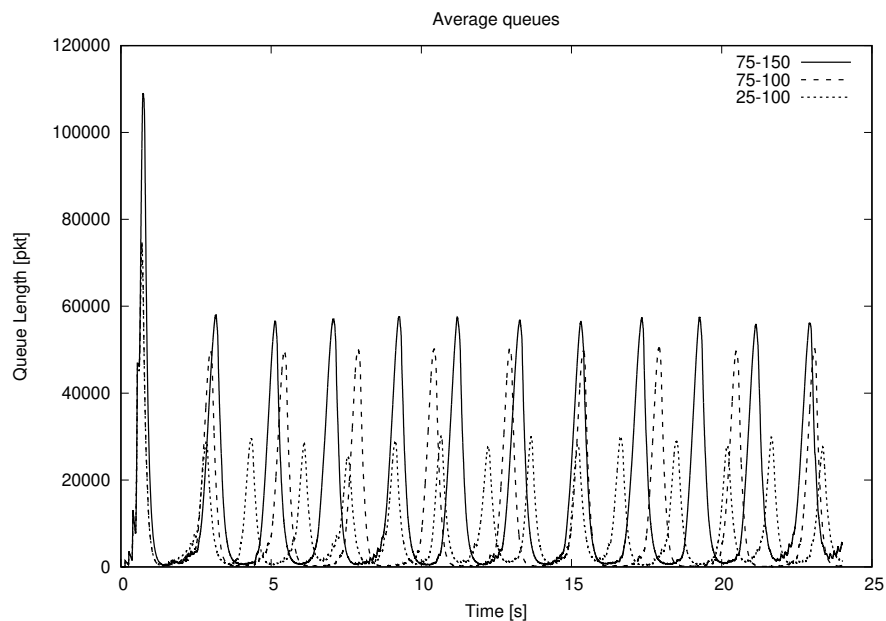


Рис. 3.2: График средней длины очереди для разных пороговых значений

Как видно из графика, увеличение диапазона между q_{\min} и q_{\max} способствует увеличению длины очереди на линке.

Для сравнений модификаций мы смоделировали сети и вывели графики очередей для всех реализованных модификаций (рис. 3.3, 3.4, 3.5, 3.6, 3.7).

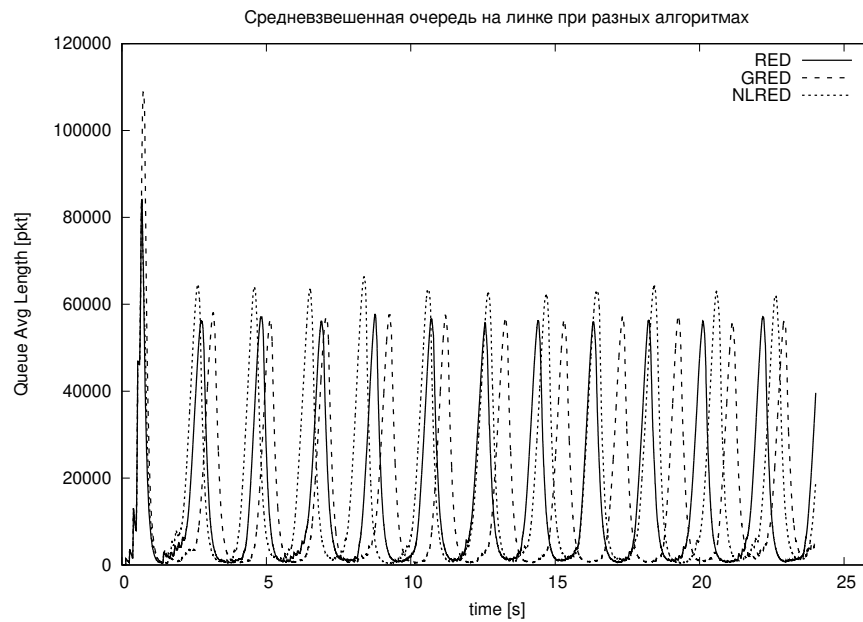


Рис. 3.3: График средневзвешанной экспоненциальной очереди для RED, GRED, NLRED

Как мы видим в 3.3, GRED и NLRED после достижения стационарного состояния имеют более модификации выдают довольно близкие значения, хотя классический алгоритм всё же более жестко отбрасывает т пакеты.

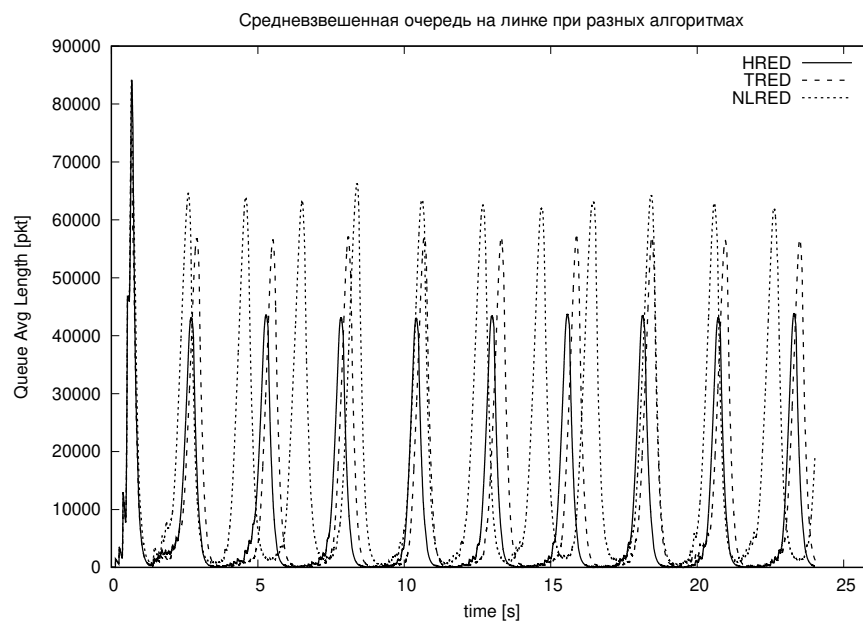


Рис. 3.4: График средневзвешанной экспоненциальной очереди для HRED, TRED, NLRED

Рассматривая нелинейные модификации 3.4, мы видим, что HRED наи-

более подходящая модификация для сетей, где потеря пакетов не является значительной проблемой, а TRED показывает результаты, по значению среднее с другими модификациями, увеличивает пропускную способность при низкой нагрузке и уменьшает задержку при высокой нагрузке.

Изучая адаптивные модификации 3.5, 3.6, 3.7. Алгоритм Feng ARED показывает наименьшую амплитуду при достижении стационарного состояния. FARED и RARED имеют одинаковый алгоритм и отличаются только двумя коэффициентами, но RARED отбрасывает гораздо меньше пакетов. Powared показывает самую большую среднюю длину очереди и из вышеперчисленных показывает наибольший разброс между данными.

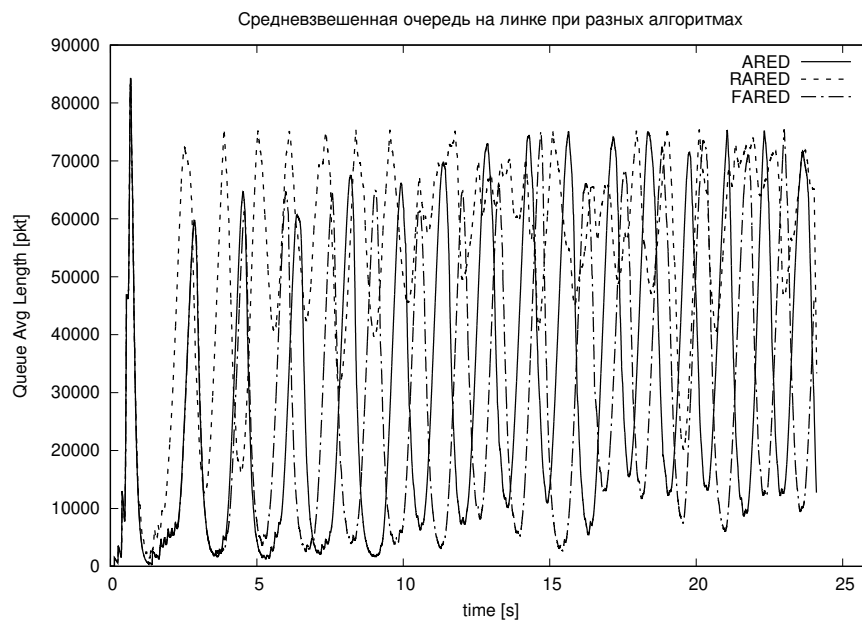


Рис. 3.5: График средневзвешенной экспоненциальной очереди для адаптивных алгоритмов

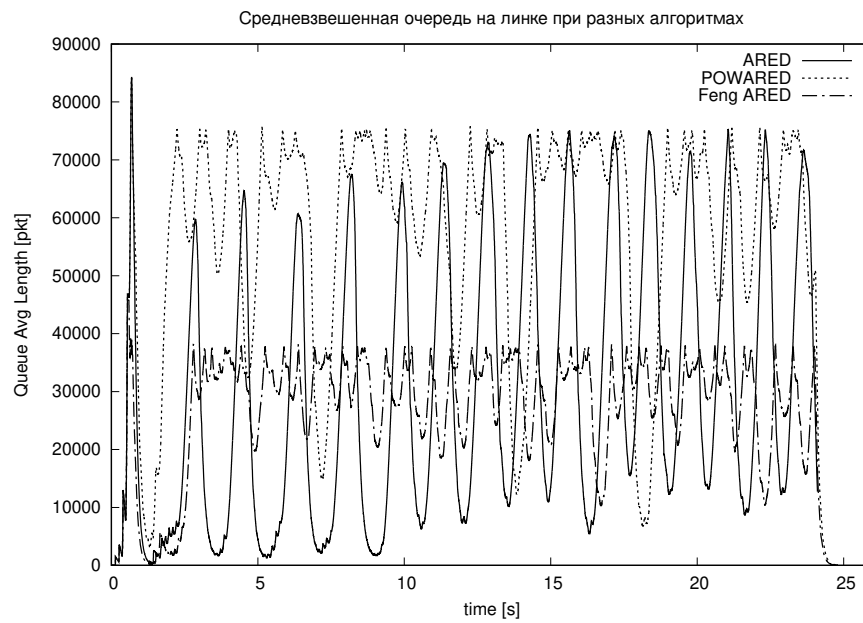


Рис. 3.6: График средневзвешенной экспоненциальной очереди для адаптивных алгоритмов

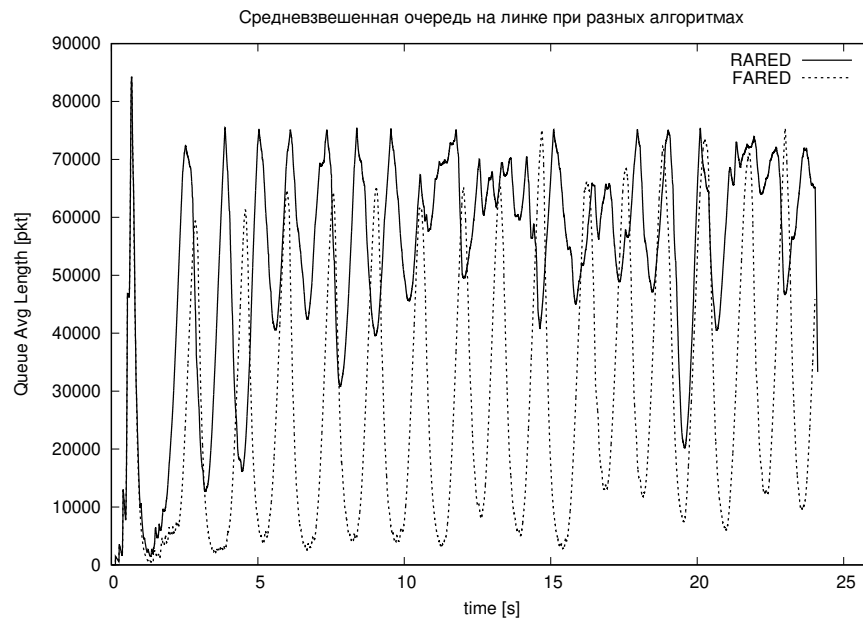


Рис. 3.7: График средневзвешенной экспоненциальной очереди для адаптивных алгоритмов

Заключение

За период практики в отделе технической поддержки пользователей (департамент технологических и информационных ресурсов) РУДН и научных центрах института прикладной математики и телекоммуникаций. были достигнуты все цели и решены все задачи, определенные в программе научной практики направления подготовки 09.03.03 «Прикладная информатика» программы «Прикладная информатика» (см. введение отчёта по практике). В процессе прохождения практики я работал с научной терминологией области исследований; научился собирать и обрабатывать данные, необходимые для формирования соответствующих выводов исследований; осуществлять целенаправленный поиск информации на русском и английском языках о новейших научных достижениях в Интернете и из других источников; строить и анализировать имитационные модели объекта исследований.

В результате прохождения данной практики я приобрел следующие практические навыки, умения, универсальные и профессиональные компетенции:

- способность управлять проектом на всех этапах его жизненного цикла (постановка задачи, планирование, реализация);
- способность составлять естественно-научные отчеты с IMRAD структурой (введение, методы и материалы, результаты и дискуссия);
- способность разрабатывать имитационные модели и проводить их анализ при решении задач в профессиональной области (составлена имитационная модель сети с алгоритмом управления очередью на маршрутизаторе типа RED);
- способность проведения работ по обработке и анализу научно-технической информации и результатов исследований (изучение необходимой литературы по теме исследования на русском и английском языках, подготовка литературного обзора по теме исследований).

Таким образом, в рамках практики я рассмотрел моделирование модуля RED с помощью программного средства NS-2 версии 2.35. Также представлена программная реализация имитационной модели сети модулем RED и проведен сравнительный анализ результатов при моделировании сети с разными входными параметрами, модификаций RED и типов TCP.

Список литературы

- [1] Floyd S. and Jacobson V. Random early detection gateways for congestion avoidance // IEEE/ACM Transactions on Networking. — 1993. — Vol. 1, no. 4. — P. 397–413.
- [2] Abouzeid Alhussein A. and Roy Sumit. Modeling random early detection in a differentiated services network // Computer Networks. — 2002. — nov. — Vol. 40, no. 4. — P. 537–556.
- [3] Hamadneh Nabhan, Al-Kasassbeh Mouhammd, Obiedat Ibrahim, and BaniKhalaf Mustafa. Revisiting the Gentle Parameter of the Random Early Detection (RED) for TCP Congestion Control // Journal of Communications. — 2019. — P. 229–235.
- [4] Hamadneh Nabhan, Murray David, Dixon Michael, and Cole Peter. Weighted RED (WTRED) Strategy for TCP Congestion Control // ICIEIS 2011 / ed. by Abd A. and et al. Manaf. — Springer. — 2011. — Vol. II of Part. — P. 421–434.
- [5] Zhou Kaiyu, Yeung Kwan L., and Li Victor O.K. Nonlinear RED: A simple yet efficient active queue management scheme // Computer Networks. — 2006. — 12. — Vol. 50, no. 18. — P. 3784–3794.
- [6] Lu Lingyun, Xiao Yang, Woo Seok, and Kim Kiseon. Nonlinear AQM for Multiple RED Routers // 2008 Third International Conference on Convergence and Hybrid Information Technology. — IEEE. — 2008. — 11.
- [7] Hu Liujia and Kshemkalyani A.D. HRED: a simple and efficient active queue management algorithm // Proceedings. 13th International Conference on Computer Communications and Networks (IEEE Cat. No.04EX969). — IEEE. — 2004.

- [8] Feng Chen-Wei, Huang Lian-Fen, Xu Cheng, and Chang Yao-Chung. Congestion Control Scheme Performance Analysis Based on Nonlinear RED // IEEE Systems Journal. — 2017. — dec. — Vol. 11, no. 4. — P. 2247–2254.
- [9] Kumhar Dharamdas, kumar Avanish, and Kewat Anil. QRED: an enhancement approach for congestion control in network communications // International Journal of Information Technology. — 2020. — oct. — Vol. 13, no. 1. — P. 221–227.
- [10] Paul Anup Kumar, Kawakami Hidehiko, Tachibana Atsuo, and Hasegawa Teruyuki. An AQM based congestion control for eNB RLC in 4G/LTE network // 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE). — IEEE. — 2016. — may.
- [11] ZHENG B. DSRED: A New Queue Management Scheme for the Next Generation Internet // IEICE Transactions on Communications. — 2006. — 3. — Vol. E89-B, no. 3. — P. 764–774.
- [12] Feng W.-C., Kandlur D.D., Saha D., and Shin K.G. A self-configuring RED gateway // 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.
- [13] Kim Tae-Hoon and Lee Kee-Hyun. Refined Adaptive RED in TCP/IP Networks // SICE-ICASE International Joint Conference 2006 Oct. 18-21, 2006 in Bexco, Busan, Korea. — IEEE. — 2006.
- [14] Ng B.K., Uddin Md. Safi, Abusin A.A.Y. Malik, and Chieng D. POWARED for Non-Linear Adaptive RED // 2005 Asia-Pacific Conference on Communications. — IEEE. — 2005.
- [15] Tahiliani Mohit P., Shet K. C., and Basavaraju T. G. FARED: Fast Adapting RED Gateways for TCP/IP Networks // IEEE/ACM Transactions on Networking. — 2012. — Vol. 1, no. 4. — P. 435–443.

Приложения

Ссылка на репозиторий: https://github.com/agsargsyan/study_2022-2023_practice

Программа симуляции

```
1 - main.tcl
2 ---
3 #Создать новый экземпляр объекта Simulator
4 set ns [new Simulator]
5
6 #Открыть трейс файл для пат, файл слишком большой, так что
   ↪ временно закомментируем
7 set nf [open output/out.nam w]
8 $ns namtrace-all $nf
9
10 #количество источников
11 set N 20
12
13 #создание узлов
14 source "nodes.tcl"
15
16 #очередь
17 source "queue.tcl"
18
19 #настройка времени моделирования
20 source "timing.tcl"
21
22 #визуализация
```

```

23 source "nam.tcl"
24
25 #процедура finish
26 source "finish.tcl"
27
28 #Запуск программы
29 $ns run
30
31
32 ...
33 - nodes.tcl
34 ...
35 set node_(r0) [$ns node] #первый маршрутизатор
36 set node_(r1) [$ns node] #второй маршрутизатор
37
38 for {set i 0} {$i < $N} {incr i} {
39     set node_(s$i) [$ns node] #источник
40     set node_(s[expr $N + $i]) [$ns node] #приемник
41 }
42
43 #линки между маршрутизаторами и другими узлами(размер буфера,
44 ↪ время, тип очереди)
45 for {set i 0} {$i < $N} {incr i} {
46     $ns duplex-link $node_(s$i) $node_(r0) 100Mb 20ms DropTail
47     $ns duplex-link $node_(s[expr $N + $i]) $node_(r1) 100Mb 20ms
48     ↪ DropTail
49 }
50
51 #линки между маршрутизаторами(размер буфера, время, тип очереди)
52 $ns simplex-link $node_(r0) $node_(r1) 20Mb 15ms RED
53 $ns simplex-link $node_(r1) $node_(r0) 15Mb 20ms DropTail
54
55 # Агенты и приложения:
56 for {set t 0} {$t < $N} {incr t} {
57     $ns color $t green

```

```

56     set tcp($t) [$ns create-connection TCP/Reno $node_(s$t) TCPSink
    ↪ $node_(s[expr $N + $t]) $t]
57     $tcp($t) set window_ 32
58     $tcp($t) set maxcwnd_ 32
59     set ftp($t) [$tcp($t) attach-source FTP]
60 }
61
62 ~ ~ ~
63
64 - nam.tcl
65 ~ ~ ~
66 #визуализация цветов, формы, расположения узлов в nam
67 $node_(r0) color "red"
68 $node_(r1) color "red"
69 $node_(r0) label "RED"
70 $node_(r1) shape "square"
71 $node_(r0) label "square"
72
73 $ns simplex-link-op $node_(r0) $node_(r1) orient right
74 $ns simplex-link-op $node_(r1) $node_(r0) orient left
75 $ns simplex-link-op $node_(r0) $node_(r1) queuePos 0
76 $ns simplex-link-op $node_(r1) $node_(r0) queuePos 0
77
78 for {set m 0} {$m < $N} {incr m} {
79     $ns duplex-link-op $node_(s$m) $node_(r0) orient right
80     $ns duplex-link-op $node_(s[expr $N + $m]) $node_(r1) orient
    ↪ left
81 }
82
83 for {set i 0} {$i < $N} {incr i} {
84     $node_(s$i) color "blue"
85     $node_(s$i) label "ftp"
86
87 }
88 ~ ~ ~

```

```

89
90 - queue.tcl
91 ---
92 #Лимит очереди
93 $ns queue-limit $node_(r0) $node_(r1) 300
94 $ns queue-limit $node_(r1) $node_(r0) 300
95
96
97 # Мониторинг размера окна TCP
98 set windowVsTime [open output/WvST w]
99 set qmon [$ns monitor-queue $node_(r0) $node_(r1) [open
   ↪ output/qm.out w]]
100 [$ns link $node_(r0) $node_(r1)] queue-sample-timeout
101
102
103 # Формирование файла с данными о размере окна TCP
104 proc plotWindow {tcpSource file} {
105     global ns
106     set time 0.01
107     set now [$ns now]
108     set cwnd [$tcpSource set cwnd_]
109     puts $file "$now $cwnd"
110     $ns at [expr $now+$time] "plotWindow $tcpSource $file"
111 }
112
113 # Мониторинг очереди:
114 set redq [$ns link $node_(r0) $node_(r1)] queue]
115 $redq set thresh_ 75
116 $redq set maxthresh_ 150
117 $redq set q_weight_ 0.002
118 $redq set linterm_ 10
119 $redq set drop-tail_ true
120
121 $redq set queue-in-bytes false
122 set tchan_ [open output/all.q w]

```

```

123 $redq trace curq_
124 $redq trace ave_
125 $redq attach $tchan_
126
127 #Для реализации разных модификаций RED,
128 $redq set gentle_ false
129
130 #$redq set nonlinear_ 1
131 #$redq set hyperbola_ 1
132 #$redq set quadratic_linear_ 1
133 #$redq set three_sections_ 1
134 #$redq set exponential_ 1
135 #$redq set smart_ 1
136 #$redq set double_slope_ 1
137
138 # Группа адаптивных алгоритмов
139 #$redq set adaptive_ 1
140 #$redq set feng_adaptive_ 1
141 #$redq set refined_adaptive_ 1
142 #$redq set fast_adaptive_ 1
143 #$redq set powared_ 1
144
145 ~ ~ ~
146
147 - timing.tcl
148 ~ ~ ~
149 #Задаем время симуляции
150 for {set r 0} {$r < $N} {incr r} {
151     $ns at 0.0 "$ftp($r) start"
152     $ns at 1.0 "plotWindow $tcp($r) $windowVsTime"
153     $ns at 24.0 "$ftp($r) stop"
154 }
155
156 $ns at 25.0 "finish"
157 ~ ~ ~

```



```

158
159 - finish.tcl
160 ---
161 #Finish procedure
162 proc finish {} {
163     global ns nf
164     $ns flush-trace
165     close $nf
166     global tchan_
167     set awkCode {
168         {#запись данных в файлы очереди и средней очереди
169         if ($1 == "Q" && NF>2) {
170             print $2, $3 >> "output/temp.q";
171             set end $2
172         }
173         else if ($1 == "a" && NF>2)
174             print $2, $3 >> "output/temp.a";
175     }
176 }
177
178 set f [open output/temp.queue w]
179 puts $f "TitleText: RED"
180 puts $f "Device: Postscript"
181
182 if { [info exists tchan_] } {
183     close $tchan_
184 }
185 #обновление данных
186 exec rm -f output/temp.q output/temp.a
187 exec touch output/temp.a output/temp.q
188
189 exec awk $awkCode output/all.q
190
191 puts $f "\"queue
192 exec cat output/temp.q >@ $f

```

```

193     puts $f \n\"ave_queue
194     exec cat output/temp.a >@ $f
195     close $f
196     # вывод в xgraph
197     exec xgraph -bb -tk -x time -t "TCPRenoCWND" output/WvsT &
198     exec xgraph -bb -tk -x time -y queue output/temp.queue &
199     exit 0
200 }

```

red.cc adaptive REDs

```

1
2 void REDQueue::updateMaxP(double new_ave, double now)
3 {
4     double part = 0.4*(edp_.th_max - edp_.th_min);
5     // AIMD rule to keep target  $Q \sim 1/2(th_{min} + th_{max})$ 
6     if ( new_ave < edp_.th_min + part && edv_.cur_max_p >
    ↪ edp_.bottom) {
7         // we increase the average queue size, so decrease max_p
8         edv_.cur_max_p = edv_.cur_max_p * edp_.beta;
9         edv_.lastset = now;
10    } else if (new_ave > edp_.th_max - part && edp_.top >
    ↪ edv_.cur_max_p ) {
11        // we decrease the average queue size, so increase max_p
12        double alpha = edp_.alpha;
13                if ( alpha > 0.25*edv_.cur_max_p )
14            alpha = 0.25*edv_.cur_max_p;
15        edv_.cur_max_p = edv_.cur_max_p + alpha;
16        edv_.lastset = now;
17    }
18 }
19
20 void REDQueue::updateMaxP_refined_adaptive(double new_ave, double
    ↪ now)
21 {

```

```

22     double part = 0.48*(edp_.th_max - edp_.th_min);
23     if ( new_ave < edp_.th_min + part && edv_.cur_max_p >
↪     edp_.bottom) {
24         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)));
25         edv_.lastset = now;
26         double maxp = edv_.cur_max_p;
27     } else if (new_ave > edp_.th_max - part && edp_.top >
↪     edv_.cur_max_p ) {
28         double alpha = edp_.alpha;
29         alpha = 0.25 * edv_.cur_max_p * ((new_ave - (edp_.th_max -
↪     part)) / (edp_.th_max - part));
30         edv_.cur_max_p = edv_.cur_max_p + alpha;
31         edv_.lastset = now;
32         double maxp = edv_.cur_max_p;
33     }
34 }
35
36
37 void REDQueue::updateMaxP_powared(double new_ave, double now)
38 {
39     double target = 0.5*(edp_.th_max + edp_.th_min);
40     int k = edp_.pwk;
41     int b = edp_.pwb;
42     int r = edp_.bf_size;
43     double v_ave = edv_.v_ave;
44
45
46
47     double delta1 = abs(pow((v_ave - target)/(b * target), k));
48     double delta2 = abs(pow((target - v_ave)/(b * (r -target)),
↪     k));
49
50     if ( new_ave < target && edv_.cur_max_p > edp_.bottom) {

```

```

51     edv_.cur_max_p = edv_.cur_max_p - delta1;
52     edv_.lastset = now;
53     double maxp = edv_.cur_max_p;
54 } else if (new_ave > target && edp_.top > edv_.cur_max_p ) {
55     edv_.cur_max_p = edv_.cur_max_p + delta2;
56     edv_.lastset = now;
57     double maxp = edv_.cur_max_p;
58 }
59 }
60
61 void REDQueue::updateMaxP_fast_adaptive(double new_ave, double
    ↪ now){
62     double part = 0.48*(edp_.th_max - edp_.th_min);
63     if ( new_ave < edp_.th_min + part && edv_.cur_max_p >
    ↪ edp_.bottom) {
64         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)));
65         edv_.lastset = now;
66         double maxp = edv_.cur_max_p;
67     } else if (new_ave > edp_.th_max - part && edp_.top >
    ↪ edv_.cur_max_p ) {
68         double alpha = edp_.alpha;
69         alpha = 0.0412 * edv_.cur_max_p * (new_ave - part) / part;
70         edv_.cur_max_p = edv_.cur_max_p + alpha;
71         edv_.lastset = now;
72         double maxp = edv_.cur_max_p;
73     }
74
75
76 }
77

```

red.cc calculate drop probability

```
1  /*
2   * Calculate the drop probability.
3   */
4  double
5  REDQueue::calculate_p_new(double v_ave, double th_max, int gentle,
6   ↪ double v_a,
7   double v_b, double v_c, double v_d, double max_p)
8  {
9   double target;
10  double exponenta = 2.7182818285;
11  double th_min = edp_.th_min;
12  double p;
13  if (gentle && v_ave >= th_max) {
14   // p ranges from max_p to 1 as the average queue
15   // size ranges from th_max to twice th_max
16   p = v_c * v_ave + v_d;
17   } else if (!gentle && v_ave >= th_max) {
18   // OLD: p continues to range linearly above max_p
19   ↪ as
20   // the average queue size ranges above th_max.
21   // NEW: p is set to 1.0
22   p = 1.0;
23   } else if (edp_.quadratic_linear == 1) {
24   target = 2 * ((th_min + th_max)/3) - th_min;
25   if(v_ave < target){
26   p = 9 * max_p * ((v_ave-th_min)/(2*(th_max-2*th_min)))
27   ↪ * ((v_ave-th_min)/(2*(th_max-2*th_min)));
28   } else if (v_ave >= target) {
29   p = max_p +
30   ↪ 3*(1-max_p)*((v_ave-target)/(th_max+th_min));
31   }
32   } else if (edp_.improved == 1) {
33   target = ((th_min + th_max)/3) + th_min;
```

```

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

```

```

58         if(v_ave < target){
59             p = a * (v_ave-th_min);
60         } else if (v_ave >= target) {
61             p = 1 - edp_.omega + b * (v_ave - target);
62         }
63     }
64     else {
65         // p ranges from 0 to max_p as the average queue
66         // size ranges from th_min to th_max
67         p = v_a * v_ave + v_b;
68         // p = (v_ave - th_min) / (th_max - th_min)
69
70         /* Added by Mohit P. Tahiliani for Nonlinear RED
↪ (NLRED) - Start */
71         if(edp_.nonlinear == 1){
72             p *= p;    // This ensures probability is a quadratic
↪ function of "average queue size" as specified in NLRED Paper
73         }
74         else if (edp_.hyperbola == 1){
75             p *= 1/p; // This ensures probability is a hyperbola
↪ function of "average queue size" as specified in HRED Paper
76         }
77         else if (edp_.exponential == 1){ // Used for RED_e
78             p = (pow(exponenta, v_ave)-pow(exponenta,
↪ th_min))/(pow(exponenta, th_max)-pow(exponenta, th_min));
79         }
80         p *= max_p;
81     }
82     if (p > 1.0)
83         p = 1.0;
84     return p;
85 }

```

red.h

```
1
2  int feng_adaptive;  /* adaptive RED: Use the Feng et al. version
   ↪ */
3  int refined_adaptive; /* Added by Mohit P. Tahliliani for
   ↪ Refined Adaptive RED (Re-ARED) */
4  int stabilized_adaptive; /* Added Stabillized Adaptive RED
   ↪ (SARED) */
5  int nonlinear;      /* Added for Nonlinear RED (NLRED) */
6  int hyperbola;      /* Added for Hyperbola RED (HRED)*/
7  int quadratic_linear; /* Added for Quadratic linear RED*/
8  int three_sections; /* Added for 3sections RED*/
9  int exponential;    /* Added for exponential RED*/
10 int improved;       /* Added for improved RED*/
11 int smart;          /* Added for smart RED*/
12 int modified;       /* Added for modified RED*/
```

ns-default.tcl

```
1
2  /*Added for new RED alghorithms*/
3
4  Queue/RED set nonlinear_ 0
5  Queue/RED set hyperbola_ 0
6  Queue/RED set quadratic_linear_ 0
7  Queue/RED set three_sections_ 0
8  Queue/RED set exponential_ 0
9  Queue/RED set improved_ 0
10 Queue/RED set smart_ 0
11 Queue/RED set modified_ 0
```


out.gp

```
1  #!/usr/bin/gnuplot -persist
2  set terminal postscript eps enhanced adobeglyphnames
3  set output "test.ps"
4  set encoding utf8
5
6  set xrange [0:26]
7
8  set terminal postscript eps
9  set output "av_queues_1GN1.eps"
10 set xlabel "Время [с]"
11 set ylabel "Длина очереди [пакеты]"
12 set title "Средневзвешенная очередь на линке при разных
   ↪ алгоритмах"
13 plot "classic.a" with lines linestyle 1 lt 1 lw 2 title "RED",
   ↪ "gentle.a" with lines linestyle 1 lt 2 lw 2 title "GRED",
   ↪ "nonlinear.a" with lines linestyle 1 lt 3 lw 2 title "NLRED"
14
15 set terminal postscript eps
16 set output "av_queues_HTN1.eps"
17 set xlabel "Время [с]"
18 set ylabel "Длина очереди [пакеты]"
19 set title "Средневзвешенная очередь на линке при разных
   ↪ алгоритмах"
20 plot "hyperbola.a" with lines linestyle 1 lt 1 lw 2 title "HRED",
   ↪ "3.a" with lines linestyle 1 lt 2 lw 2 title "TRED",
   ↪ "nonlinear.a" with lines linestyle 1 lt 3 lw 2 title "NLRED"
```