

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Бизнес-проект «Разработка блока анализа
темпорально-корреляционных свойств
сетевого трафика»**

Студент


(подпись, дата)

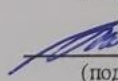
Братчиков Александр Андреевич
(ФИО полностью)

Руководитель


(подпись, дата)

Р.А. Бикташев
(фамилия, инициалы)

Нормоконтролёр


(подпись, дата)

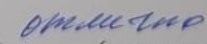
А.С. Бычков
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

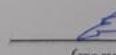
Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой  (протокол заседания ГЭК от 01.06.2026 № 21)

Секретарь ГЭК


(подпись)

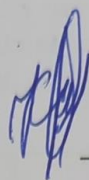
Е.И. Гурин
(фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»



«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» апреля 2026г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Братчиков Александр Андреевич гр. 22ВВП2 факультета ФВТ
2. Тема работы «Бизнес-проект «Разработка блока анализа темпорально-корреляционных свойств сетевого трафика»

Тема утверждена приказом ПГУ №347 от "07" апреля 2026 г.

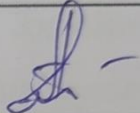
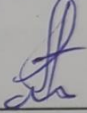
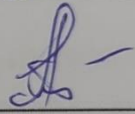
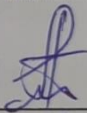
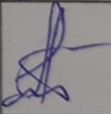
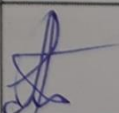
3. Руководитель бакалаврской работы к.т.н., доцент, каф. ВТ ПГУ Бикташев Р.А

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать и реализовать блок анализа темпорально-корреляционных свойств трафика, обладающего плохой сходимостью выборочных характеристик качества обслуживания. Исходными данными для работы блока является входной поток разнородного трафика, обладающего устойчивым распределением автокорреляционных характеристик. Продукт должен представлять собой программно-аппаратное обеспечение в виде блока анализа темпорально-корреляционных свойств трафика, обладающего плохой сходимостью выборочных характеристик качества обслуживания, обеспечивающего улучшение характеристик QoS/QoE по сравнению с известными реализациями сетевого оборудования L2/L3-уровней. Этот блок может быть реализован как в виде программного комплекса, предназначенного под известные реализации сетевого оборудования, либо в виде аппаратного ускорителя под известные открытые архитектуры. Кроме того, продукт должен обладать возможностями применения в условиях крайне ненадёжной сети, например в рамках IoT и MQTT-ориентированного оборудования

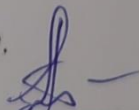
5. Перечень подлежащих разработке вопросов Требуется провести анализ предметных областей: изучить возможности инструментальной среды моделирования сетевой активности, например GNS-3. Изучить основные алгоритмы доступа к критическим ресурсам и их автоматные модели, а также алгоритмы диспетчеризации очередей в сетевом оборудовании с поддерживаемой коммутацией пакетов. Изучить известные принципы построения блоков дискриминаторов в контексте корреляционного анализа с целью дальнейшей коррекции показателей QoS/QoE. Выбор программных образов с целью соблюдения баланса между адекватностью оценок инструментальными средствами сетевого симулятора временных характеристик вычислительной сети и количественными затратами вычислительных ресурсов. Определить архитектуру блока анализа темпорально-корреляционных свойств трафика. Изучить функционал библиотек, связанных со статистической, частотной, частотно-временной и прочей субматематической обработкой входных данных, например GSL.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	20	07.04.2026-10.04.2026	
2	Анализ требований	32	11.04.2026-15.04.2026	
3	Выбор программных образов для симуляции вычислительной сети	30	16.04.2026-20.04.2026	
4	Разработка математической модели дискриминатора	80	21.04.2026-01.05.2026	
5	Разработка алгоритма активного управления джиттером	18	02.05.2026-06.05.2026	
6	Сборка тестового стенда	40	07.05.2026-10.05.2026	
7	Отладка и тестирование	50	11.05.2026-15.05.2026	
8	Оформление пояснительной записки	54	16.05.2026-21.05.2026	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы



Бикташев Р.А.

(подпись, дата)

Задание к исполнению принял студент

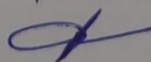


Братчиков А.А.

(подпись, дата)

Работу к защите допустить

Декан факультета



Фионова Л.Р.

(подпись, дата)

02.06.2026

Реферат

Бакалаврская работа содержит 142 страницы, 26 рисунков, 4 таблицы, 10 источников, 3 приложения

АНАЛИЗ ТРАФИКА, АКТИВНОЕ УПРАВЛЕНИЕ ОЧЕРЕДЬЮ, ДЖИТТЕР, ВРЕМЕННЫЕ РЯДЫ, ARIMA, BASH, C++, C, LINUX, RAW-SOCKET

Цель работы – разработка и реализация блока анализа темпорально-корреляционных свойств сетевого трафика, обеспечивающего прогнозирование межпакетных интервалов и адаптивное управление очередью для минимизации джиттера.

Объектом разработки является программный блок анализа временных и корреляционных характеристик трафика, выполненный в виде динамически загружаемых модулей. Блок интегрируется в состав программного маршрутизатора и позволяет осуществлять сбор статистики интервалов между пакетами, построение и оценку модели ARIMA(2,1,1), прогнозирование момента поступления следующего пакета и адаптивную коррекцию racing-интервала.

Разработка проводилась в текстовом редакторе Visual Studio Code. В качестве языка программирования был выбран C++ с использованием библиотеки GNU Scientific Library для реализации методов линейной алгебры.

В результате работы разработан блок анализа темпорально-корреляционных свойств трафика, включающий сбор межпакетных интервалов, оценку коэффициентов ARIMA(2,1,1) методом Юла-Уокера, прогнозирование и адаптивное управление racing-интервалом. Блок интегрирован в программный маршрутизатор с модульной загрузкой планировщиков, конфигурированием через INI-файл и системой логирования..

					ПГУ1.090301.31.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Братчиков А.А.		01.06.20	Бизнес-проект «Разработка блока анализа темпорально-корреляционных свойств сетевого трафика» Пояснительная записка	Лит.	Лист	Листов
Провер.		Бикташев Р.А.		01.06.20			2	
Н. Контр		Бычков А.С.		01.06.20		ФВТ, каф. ВТ		
Утверд.		Митрохин М.А.		01.06.20		гр. 22ВВП2		

Содержание

Перечень сокращений и обозначений	6
Введение	7
1 Разработка блока анализа темпорально-корреляционных свойств сетевого трафика	8
1.1 Актуальность	8
1.2 Анализ аналогов	10
1.2.1 Cisco IOS / IOS-XE	13
1.2.2 MikroTik RouterOS	18
1.3 Функциональное сравнение с аналогами.....	21
1.4 Постановка задачи.....	24
1.5 Выбор инструментов разработки	24
1.5.1 Язык программирования	24
1.5.2 Среда разработки.....	26
1.5.3 Среда моделирования сетей.	27
1.6 Настройка инструментов разработки.....	29
1.7 Архитектура разрабатываемой системы.....	33
1.8 Разработка маршрутизатора и модулей очереди	39
1.8.1 Разработка классов.....	39
1.8.2 Класс Packet	41
1.8.3 Класс RawSocket.....	41
1.8.4 Класс Receiver.....	42
1.8.5 Класс Sender.....	42
1.8.6 Класс IScheduler.....	42
1.8.7 Класс SchedulerLoader.....	43
1.8.8 Класс DirectionWorker.....	43
1.8.9 Класс Router	43
1.8.10 Класс Logger	43
1.8.11 Класс ConfigLoader	44

1.8.10 Класс FifoScheduler	44
1.8.11 Класс RedScheduler	45
1.8.12 Класс CodelScheduler	45
1.8.13 Класс ArimaScheduler.....	45
1.9 Тестирование и отладка.....	46
1.9.1 Тестирование FIFO.....	48
1.9.2 Тестирование RED	50
1.9.3 Тестирование CoDel.....	51
1.9.4 Тестирование ARIMA.....	53
2 Охрана труда.....	56
2.1 Описание негативных факторов, воздействующих на разработчика и способы их устранения.....	56
2.1.1 Организация рабочего места.....	56
2.1.2 Пожарная безопасность	57
2.1.3 Электробезопасность	58
2.1.4 Режим труда и отдыха	58
3 Стартап проект.....	60
3.1 Описание существующей проблемы на рынке	60
3.2 Описание решения проблемы и продуктового решения.....	62
3.3 Объём, структура и развитие рынка	66
3.4 Анализ конкурентов.....	68
3.5 Бизнес-модель проекта	72
3.7 Описание рисков	74
Заключение	76
Список использованных источников	77
Приложение А	78
Приложение А.1 – UML-диаграмма классов.....	79
Приложение А.2 – UML-диаграмма прецедентов	80
Приложение А.3 – UML-диаграмма последовательностей	81

Приложение А.4 – UML-диаграмма развёртывания	82
Приложение Б – Листинг программы	82
Приложение С – Презентация.....	134

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		5

Перечень сокращений и обозначений

В данной работе используются следующие сокращения

ЭВМ — электронно-вычислительная машина

API (Application Programming Interface) — программный интерфейс приложения

ARIMA (AutoRegressive Integrated Moving Average) — авторегрессионное интегрированное скользящее среднее

CLI (Command Line Interface) — интерфейс командной строки

CoDel (Controlled Delay) — контролируемая задержка

DPDK (Data Plane Development Kit) — комплект разработки для высокоскоростной обработки сетевых пакетов

FARIMA (Fractional ARIMA) — фрактальное авторегрессионное интегрированное скользящее среднее

FIFO (First In, First Out) — «первым пришёл — первым обслужен»

GSL (GNU Scientific Library) — научная библиотека GNU

IIoT (Industrial Internet of Things) — промышленный интернет вещей

IOS (Internetwork Operating System) — межсетевая операционная система (Cisco)

MQTT (Message Queuing Telemetry Transport) — протокол передачи телеметрии с очередями сообщений

QoS (Quality of Service) — качество обслуживания

RED (Random Early Detection) — случайное раннее обнаружение

UML (Unified Modeling Language) — унифицированный язык моделирования

Введение

В настоящее время наблюдается стремительный рост объёмов сетевого трафика, обусловленный развитием цифровых технологий, распространением облачных сервисов, видеостриминга и приложений реального времени. Увеличение нагрузки на сетевую инфраструктуру приводит к ужесточению требований к качеству обслуживания (QoS), включая такие параметры, как задержка, потери пакетов и, в особенности, джиттер. Стабильность этих характеристик становится критически важной для корректной работы современных сетевых сервисов. Одним из перспективных направлений анализа сетевого трафика является использование математических моделей временных рядов, таких как ARIMA и FARIMA. Данные модели позволяют учитывать временные зависимости и прогнозировать поведение трафика, что открывает возможности для более эффективного управления сетевыми ресурсами.

Однако, несмотря на высокую эффективность данных методов, их практическое применение в системах управления сетевым трафиком остаётся ограниченным. В большинстве существующих решений используются реактивные подходы, не учитывающие временную структуру данных.

В связи с этим возникает необходимость разработки решений, направленных на интеграцию методов анализа временных рядов в системы обработки сетевого трафика. Создание модели, способной учитывать темпоральные и корреляционные свойства трафика и применять их для снижения джиттера, является актуальной задачей, имеющей как научное, так и практическое значение.

Настоящая работа направлена на разработку и исследование модели анализа сетевого трафика, основанной на применении ARIMA-подхода, с целью повышения качества обслуживания и снижения вариации задержек в сети.

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

1 Разработка блока анализа темпорально-корреляционных свойств сетевого трафика

1.1 Актуальность

Актуальность темы исследования обусловлена стремительным развитием сетевых технологий и постоянным ростом объёмов передаваемых данных, а также увеличением числа сервисов, предъявляющих высокие требования к качеству передачи информации. В современных условиях цифровизации особое значение приобретают приложения реального времени, такие как системы видеоконференцсвязи, IP-телефония, потоковое мультимедиа, онлайн-игры и распределённые интерактивные сервисы. Для корректного функционирования подобных систем критически важны не только пропускная способность сети и средняя задержка передачи данных, но и стабильность этих параметров во времени.

Одним из ключевых показателей качества сетевого обслуживания (QoS) является джиттер — вариация задержки доставки пакетов. В отличие от средней задержки, джиттер отражает нестабильность сети и напрямую влияет на воспринимаемое качество сервисов. Даже при относительно небольшой средней задержке значительные колебания времени доставки пакетов могут приводить к искажению аудио- и видеопотоков, потере синхронизации, буферизации и другим негативным эффектам. Таким образом, проблема минимизации джиттера является одной из центральных задач при обеспечении качества обслуживания в сетях передачи данных.

Сложность данной задачи обусловлена высокой динамичностью сетевого трафика. Нагрузки в сети могут изменяться во времени, формируя сложные, нелинейные и зачастую стохастические зависимости между различными параметрами трафика, такими как интенсивность потока, задержки, потери пакетов и пропускная способность каналов. В таких условиях традиционные методы управления трафиком, ориентированные на статические или усреднённые

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

характеристики, оказываются недостаточно эффективными, поскольку не учитывают временную структуру и взаимосвязи между параметрами.

В этой связи особую актуальность приобретает анализ темпоральных свойств сетевого трафика, позволяющий учитывать изменение характеристик во времени. Изучение временных рядов задержек и других параметров открывает возможности для выявления закономерностей, трендов и периодичности, а также для построения моделей прогнозирования поведения сети. Дополнительное значение имеет исследование корреляционных зависимостей между параметрами трафика, что позволяет выявить скрытые взаимосвязи и факторы, влияющие на формирование джиттера.

Применение методов темпорально-корреляционного анализа позволяет перейти от реактивных подходов управления сетью к проактивным, основанным на прогнозировании и предотвращении нежелательных эффектов. Это, в свою очередь, создаёт предпосылки для разработки интеллектуальных систем управления трафиком, способных адаптироваться к изменяющимся условиям и обеспечивать более стабильное качество передачи данных.

Несмотря на наличие значительного числа исследований в области управления сетевым трафиком и обеспечения QoS, задача снижения джиттера остаётся недостаточно решённой, особенно в условиях высоконагруженных и гетерогенных сетей. Многие существующие решения ориентированы на узкие сценарии применения или требуют значительных вычислительных ресурсов, что ограничивает их практическое использование. Кроме того, недостаточно изучены возможности применения комплексного анализа временных и корреляционных характеристик трафика для повышения стабильности задержек.

Важным направлением исследований является использование моделирования и имитационных подходов для анализа поведения сетевых систем. Моделирование позволяет воспроизводить различные сценарии нагрузки, исследовать влияние параметров на характеристики сети и оценивать эффективность предлагаемых решений без необходимости вмешательства в

					ПГУ1.090301.31.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

реальные инфраструктуры. Это особенно важно на этапе разработки новых алгоритмов и методов, направленных на оптимизацию сетевых процессов.

Таким образом, разработка и исследование блока, основанного на анализе темпорально-корреляционных свойств сетевого трафика и направленной на снижение джиттера, представляет собой актуальную научно-техническую задачу. Решение данной задачи позволит повысить стабильность сетевых задержек, улучшить качество обслуживания пользователей и расширить возможности применения современных сетевых технологий в критически важных и чувствительных к задержкам приложениях.

Актуальность рассматриваемого блока также определяется его практической значимостью, поскольку результаты исследования ориентированы на разработку универсального модуля анализа трафика, который может быть интегрирован в состав сетевого оборудования или программно-аппаратных комплексов. Полученные решения могут быть использованы при проектировании и оптимизации сетевых систем, а также при создании встроенных средств мониторинга и управления трафиком. Это делает данное направление востребованным как в научной, так и в прикладной сфере.

1.2 Анализ аналогов

В рамках данной выпускной квалификационной работы требуется провести сравнительный анализ аналогов приведенного выше блока. Поскольку данный блок является составной частью более крупных систем обеспечения QoS, целесообразно проводить сравнение по тем функциональным возможностям, которые реализуются или модифицируются данным блоком, в сопоставлении с аналогичными решениями. Сравнительный анализ выполняется на основе критериев функциональности, производительности, гибкости настройки. Проведение данного этапа необходимо для последующего корректного формирования функциональных требований к разрабатываемому модулю.

Результаты сравнительного анализа аналогов по выбранным критериям приведены в таблице 1.

					ПГУ1.090301.31.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 1 – Сравнение аналогов по критериям

ПО маршрутизатора	Функциональные возможности	Производительность (джиттер при скорости 500Кб/с)	Сложность использования и развертывания
Cisco IOS / IOS-XE	Широкий набор функций: QoS (CBWFQ, LLQ), ACL, NAT, MPLS, VPN, гибкая маршрутизация, поддержка большого числа протоколов	6.331 мс	Высокая, определяется сложностью командного интерфейса и большим количеством настраиваемых параметров
MikroTik RouterOS	Базовый и расширенный QoS (queues, simple queues, queue tree), firewall, NAT, VPN, routing, удобный GUI (WinBox)	0.071 мс	Низкая обусловлена наличием графических инструментов настройки и относительно простой логикой конфигурации

Производительность будет оцениваться в виртуальной среде GNS3 с использованием набора команд, моделирующих сетевую нагрузку. Для генерации трафика будет применяться iPerf на втором и третьем узлах, а также выполняться проверка задержек с первого узла на третий с помощью утилиты ping. Схема стенда представлена на рисунке 1.

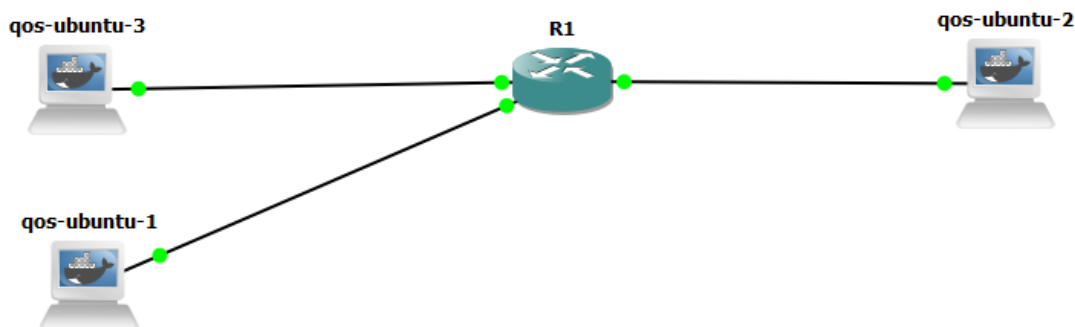


Рисунок 1 – схема стенда для тестирования производительности QoS ПО

Также выполнена настройка IP-адресации и таблиц маршрутизации на конечных узлах тестового стенда. Конфигурация конечных узлов представлена на рисунке 2.

10.11 Qt 5.15.2 and PyQt 5.15.11.

loaded, you may have to be registered with the vendor in order to download the file.
ided, you may have to be registered with the vendor in order to download the file.

```

root@qos-ubuntu-2: /
qos-ubuntu-2 console is now available... Press RETURN to get started.
root@qos-ubuntu-2: /# ip addr add 10.0.3.2/24 dev eth0
root@qos-ubuntu-2: /# ip link set eth0 up
root@qos-ubuntu-2: /# ip route add default via 10.0.3.1
root@qos-ubuntu-2: /# ping 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data:
64 bytes from 10.0.1.2: icmp_seq=1 ttl=63 time=38.2 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=63 time=26.8 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=63 time=15.4 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=63 time=27.2 ms
64 bytes from 10.0.1.2: icmp_seq=5 ttl=63 time=23.9 ms
^C
--- 10.0.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 443ms
rtt min/avg/max/mdev = 15.428/26.308/38.220/7.310 ms
root@qos-ubuntu-2: /#

root@qos-ubuntu-3: /
qos-ubuntu-3 console is now available... Press RETURN to get started.
root@qos-ubuntu-3: /# ip addr add 10.0.1.2/24 dev eth0
root@qos-ubuntu-3: /# ip link set eth0 up
root@qos-ubuntu-3: /# ip route add default via 10.0.1.1
root@qos-ubuntu-3: /# ping 10.0.3.1
PING 10.0.3.1 (10.0.3.1) 56(84) bytes of data:
64 bytes from 10.0.3.1: icmp_seq=1 ttl=255 time=19.2 ms
64 bytes from 10.0.3.1: icmp_seq=2 ttl=255 time=5.55 ms
64 bytes from 10.0.3.1: icmp_seq=3 ttl=255 time=6.43 ms
64 bytes from 10.0.3.1: icmp_seq=4 ttl=255 time=7.54 ms
64 bytes from 10.0.3.1: icmp_seq=5 ttl=255 time=5.36 ms
64 bytes from 10.0.3.1: icmp_seq=6 ttl=255 time=1.85 ms
64 bytes from 10.0.3.1: icmp_seq=7 ttl=255 time=12.1 ms
^C
--- 10.0.3.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 666ms
rtt min/avg/max/mdev = 1.853/8.292/19.193/5.278 ms
root@qos-ubuntu-3: /#

root@qos-ubuntu-1: /
qos-ubuntu-1 console is now available... Press RETURN to get started.
root@qos-ubuntu-1: /# ip addr add 10.0.2.2/24 dev eth0
root@qos-ubuntu-1: /# ip link set eth0 up
root@qos-ubuntu-1: /# ip route add default via 10.0.2.1
root@qos-ubuntu-1: /# ping 10.0.3.2
PING 10.0.3.2 (10.0.3.2) 56(84) bytes of data:
64 bytes from 10.0.3.2: icmp_seq=1 ttl=63 time=32.5 ms
64 bytes from 10.0.3.2: icmp_seq=2 ttl=63 time=22.7 ms
64 bytes from 10.0.3.2: icmp_seq=3 ttl=63 time=23.6 ms
64 bytes from 10.0.3.2: icmp_seq=4 ttl=63 time=24.3 ms
64 bytes from 10.0.3.2: icmp_seq=5 ttl=63 time=24.8 ms
^C
--- 10.0.3.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 443ms
rtt min/avg/max/mdev = 22.657/25.552/32.487/3.538 ms
root@qos-ubuntu-1: /#
  
```

Рисунок 2 – Конфигурация конечных узлов стенда

1.2.1 Cisco IOS / IOS-XE

Cisco IOS – сетевая операционная система, широко применяемая в маршрутизаторах и коммутаторах Cisco. Обладает развитым набором инструментов QoS и позволяет реализовывать сложные политики обработки трафика в корпоративных и операторских сетях. Благодаря высокой зрелости и распространённости, система фактически является отраслевым стандартом.

Однако, как и любое программное обеспечение, Cisco IOS имеет свои особенности.

Проведем анализ по следующим параметрам:

– удобство конфигурации и управления. Настройка QoS в Cisco IOS осуществляется через CLI с использованием class-map, policy-map и service-policy. Такой подход обеспечивает высокую гибкость, но требует значительного опыта. Конфигурации могут становиться громоздкими и трудночитаемыми, особенно в сложных сценариях;

– гибкость интеграции с другими технологиями. Cisco IOS хорошо интегрируется с другими сетевыми технологиями Cisco (MPLS, VPN, VLAN, NetFlow). Это позволяет строить комплексные решения с единым управлением политиками QoS. В этом аспекте система демонстрирует высокий уровень совместимости;

– поддержка мониторинга и диагностики. Система предоставляет развитые инструменты мониторинга (show-команды, NetFlow, SNMP), позволяющие анализировать состояние очередей, загрузку каналов и эффективность QoS-политик. Однако интерпретация этих данных требует опыта;

– производительность при высокой нагрузке. Cisco IOS обеспечивает стабильную работу под нагрузкой, особенно на специализированном оборудовании Cisco. QoS-механизмы эффективно работают на аппаратном уровне (ASIC), что снижает нагрузку на CPU. Тем не менее, производительность напрямую зависит от модели устройства.

Для проведения нагрузочного тестирования производительности выполнена настройка сетевых интерфейсов стенда и подключение маршрутизатора Cisco к эмулируемой сети. Настройка конфигурации маршрутизатора приведена на рисунке 3.

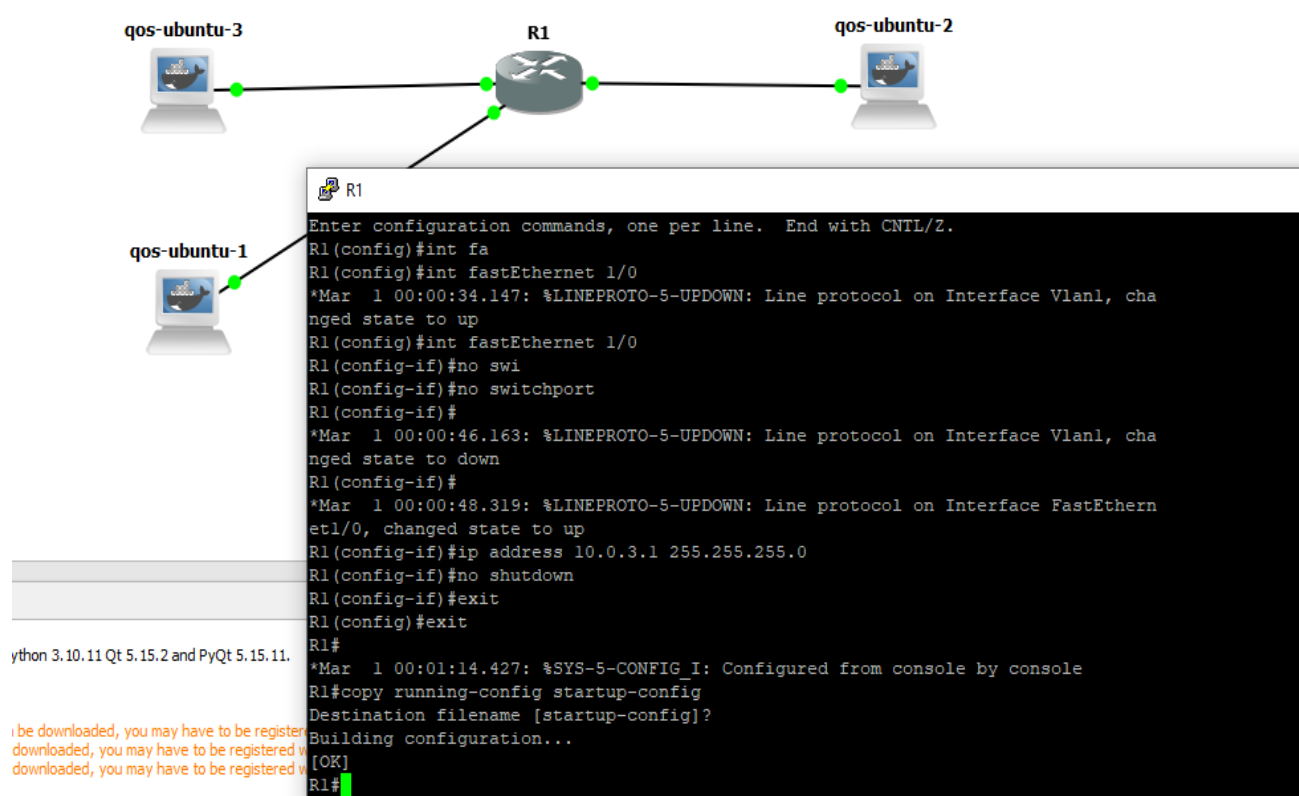


Рисунок 3 – Конфигурация Cisco

Для оценки пропускной способности и джиттера выполнена генерация трафика между узлами qos-ubuntu-1 и qos-ubuntu-3 с помощью утилиты iperf3. Результаты измерений представлены на рисунке 4.

Одновременно с генерацией трафика выполнялась проверка задержки командой ping между узлами qos-ubuntu-1 и qos-ubuntu-3. Результаты показали значительный рост средней задержки в условиях высокой загрузки сети. Измеренные значения задержки приведены на рисунке 5

```

root@qos-ubuntu-2: /
[ 12] local 10.0.3.2 port 5201 connected to 10.0.1.2 port 60940
[ 14] local 10.0.3.2 port 5201 connected to 10.0.1.2 port 60944
[ 16] local 10.0.3.2 port 5201 connected to 10.0.1.2 port 60954
[ 18] local 10.0.3.2 port 5201 connected to 10.0.1.2 port 60956
[ 20] local 10.0.3.2 port 5201 connected to 10.0.1.2 port 60960
[ 22] local 10.0.3.2 port 5201 connected to 10.0.1.2 port 60964
[ 24] local 10.0.3.2 port 5201 connected to 10.0.1.2 port 60966
[ ID] Interval          Transfer      Bitrate
[  5] 0.00-69.05 sec    256 KBytes   30.4 Kbits/sec
[  8] 0.00-69.05 sec    128 KBytes   15.2 Kbits/sec
[ 10] 0.00-69.05 sec    128 KBytes   15.2 Kbits/sec
[ 12] 0.00-69.05 sec    128 KBytes   15.2 Kbits/sec
[ 14] 0.00-69.05 sec    128 KBytes   15.2 Kbits/sec
[ 16] 0.00-69.05 sec    128 KBytes   15.2 Kbits/sec
[ 18] 0.00-69.05 sec    107 KBytes   12.8 Kbits/sec
[ 20] 0.00-69.05 sec    107 KBytes   12.8 Kbits/sec
[ 22] 0.00-69.05 sec    107 KBytes   12.8 Kbits/sec
[ 24] 0.00-69.05 sec    107 KBytes   12.8 Kbits/sec
[SUM] 0.00-69.05 sec  1.29 MBytes   157 Kbits/sec
iperf3: error - select failed: Bad file descriptor
-----
Server listening on 5201
-----

root@qos-ubuntu-3: /
 7] 8.00-9.00 sec 0.00 Bytes 0.00 bits/sec 0 52.3 KBytes
 9] 8.00-9.00 sec 0.00 Bytes 0.00 bits/sec 0 53.7 KBytes
11] 8.00-9.00 sec 0.00 Bytes 0.00 bits/sec 0 48.1 KBytes
13] 8.00-9.00 sec 0.00 Bytes 0.00 bits/sec 0 48.1 KBytes
15] 8.00-9.00 sec 93.3 KBytes 765 Kbits/sec 0 29.7 KBytes
17] 8.00-9.00 sec 31.1 KBytes 255 Kbits/sec 0 14.1 KBytes
19] 8.00-9.00 sec 31.1 KBytes 255 Kbits/sec 0 14.1 KBytes
21] 8.00-9.00 sec 0.00 Bytes 0.00 bits/sec 0 17.0 KBytes
23] 8.00-9.00 sec 31.1 KBytes 255 Kbits/sec 0 15.6 KBytes
SUM] 8.00-9.00 sec 696 KBytes 5.70 Mbits/sec 0
-----
 5] 9.00-10.00 sec 382 KBytes 3.13 Mbits/sec 0 195 KBytes
 7] 9.00-10.00 sec 0.00 Bytes 0.00 bits/sec 0 52.3 KBytes
 9] 9.00-10.00 sec 0.00 Bytes 0.00 bits/sec 0 53.7 KBytes
11] 9.00-10.00 sec 0.00 Bytes 0.00 bits/sec 0 48.1 KBytes
13] 9.00-10.00 sec 0.00 Bytes 0.00 bits/sec 0 48.1 KBytes
15] 9.00-10.00 sec 0.00 Bytes 0.00 bits/sec 0 29.7 KBytes
17] 9.00-10.00 sec 0.00 Bytes 0.00 bits/sec 0 17.0 KBytes
19] 9.00-10.00 sec 0.00 Bytes 0.00 bits/sec 0 17.0 KBytes
21] 9.00-10.00 sec 0.00 Bytes 0.00 bits/sec 0 19.8 KBytes
23] 9.00-10.00 sec 0.00 Bytes 0.00 bits/sec 0 18.4 KBytes
SUM] 9.00-10.00 sec 382 KBytes 3.13 Mbits/sec 0

```

Рисунок 4 – Загруженность сети

```

root@qos-ubuntu-1:/# ping 10.0.3.2
PING 10.0.3.2 (10.0.3.2) 56(84) bytes of data.
64 bytes from 10.0.3.2: icmp_seq=1 ttl=63 time=2542 ms
64 bytes from 10.0.3.2: icmp_seq=2 ttl=63 time=1801 ms
64 bytes from 10.0.3.2: icmp_seq=3 ttl=63 time=1523 ms
64 bytes from 10.0.3.2: icmp_seq=4 ttl=63 time=1221 ms
64 bytes from 10.0.3.2: icmp_seq=5 ttl=63 time=1640 ms
64 bytes from 10.0.3.2: icmp_seq=6 ttl=63 time=2634 ms
64 bytes from 10.0.3.2: icmp_seq=7 ttl=63 time=3070 ms
64 bytes from 10.0.3.2: icmp_seq=8 ttl=63 time=3739 ms
64 bytes from 10.0.3.2: icmp_seq=9 ttl=63 time=4146 ms
64 bytes from 10.0.3.2: icmp_seq=10 ttl=63 time=3807 ms
64 bytes from 10.0.3.2: icmp_seq=11 ttl=63 time=4316 ms
64 bytes from 10.0.3.2: icmp_seq=12 ttl=63 time=3446 ms
64 bytes from 10.0.3.2: icmp_seq=13 ttl=63 time=2465 ms
64 bytes from 10.0.3.2: icmp_seq=14 ttl=63 time=1574 ms
64 bytes from 10.0.3.2: icmp_seq=15 ttl=63 time=1906 ms
64 bytes from 10.0.3.2: icmp_seq=16 ttl=63 time=2579 ms
64 bytes from 10.0.3.2: icmp_seq=17 ttl=63 time=1822 ms
64 bytes from 10.0.3.2: icmp_seq=18 ttl=63 time=2092 ms
64 bytes from 10.0.3.2: icmp_seq=19 ttl=63 time=1730 ms
64 bytes from 10.0.3.2: icmp_seq=20 ttl=63 time=1421 ms
64 bytes from 10.0.3.2: icmp_seq=21 ttl=63 time=1189 ms
64 bytes from 10.0.3.2: icmp_seq=22 ttl=63 time=1071 ms
64 bytes from 10.0.3.2: icmp_seq=23 ttl=63 time=1065 ms
64 bytes from 10.0.3.2: icmp_seq=24 ttl=63 time=1295 ms
64 bytes from 10.0.3.2: icmp_seq=25 ttl=63 time=1663 ms
64 bytes from 10.0.3.2: icmp_seq=26 ttl=63 time=963 ms
64 bytes from 10.0.3.2: icmp_seq=27 ttl=63 time=1332 ms
64 bytes from 10.0.3.2: icmp_seq=28 ttl=63 time=1667 ms
64 bytes from 10.0.3.2: icmp_seq=29 ttl=63 time=985 ms
64 bytes from 10.0.3.2: icmp_seq=30 ttl=63 time=1493 ms
64 bytes from 10.0.3.2: icmp_seq=31 ttl=63 time=1355 ms
64 bytes from 10.0.3.2: icmp_seq=32 ttl=63 time=1223 ms
64 bytes from 10.0.3.2: icmp_seq=33 ttl=63 time=1193 ms
64 bytes from 10.0.3.2: icmp_seq=34 ttl=63 time=1468 ms
64 bytes from 10.0.3.2: icmp_seq=35 ttl=63 time=519 ms
64 bytes from 10.0.3.2: icmp_seq=36 ttl=63 time=20.7 ms
64 bytes from 10.0.3.2: icmp_seq=37 ttl=63 time=18.1 ms
64 bytes from 10.0.3.2: icmp_seq=38 ttl=63 time=16.3 ms
64 bytes from 10.0.3.2: icmp_seq=39 ttl=63 time=27.9 ms
^C
--- 10.0.3.2 ping statistics ---
39 packets transmitted, 39 received, 0% packet loss, time 40870ms
rtt min/avg/max/mdev = 16.303/1744.428/4315.519/1081.083 ms, pipe 5

```

Рисунок 5 – Задержка ICMP пакетов при загруженной сети

Дополнительно проведено тестирование UDP-трафика между узлами qos-ubuntu-1 и qos-ubuntu-3. Ввиду ограниченной производительности используемого маршрутизатора Cisco 3745 целевая скорость генерации трафика была выбрана

равной 500 Кбит/с. Результаты тестирования при заданной скорости представлены на рисунке 6

```

Server listening on 5201
-----
Accepted connection from 10.0.1.10, port 37974
[ 5] local 10.0.2.10 port 5201 connected to 10.0.1.10 port 53987
[ ID] Interval          Transfer      Bitrate      Jitter      Lost/Total Datagrams
[ 5]  0.00-1.00    sec  56.6 KBytes  463 Kbits/sec  5.784 ms    0/40 (0%)
[ 5]  1.00-2.00    sec  62.2 KBytes  510 Kbits/sec  6.360 ms    0/44 (0%)
[ 5]  2.00-3.00    sec  60.8 KBytes  498 Kbits/sec  6.421 ms    0/43 (0%)
[ 5]  3.00-4.00    sec  60.8 KBytes  498 Kbits/sec  6.254 ms    0/43 (0%)
[ 5]  4.00-5.00    sec  60.8 KBytes  498 Kbits/sec  6.134 ms    0/43 (0%)
[ 5]  5.00-6.00    sec  60.8 KBytes  498 Kbits/sec  6.498 ms    0/43 (0%)
[ 5]  6.00-7.00    sec  60.8 KBytes  498 Kbits/sec  6.374 ms    0/43 (0%)
[ 5]  7.00-8.00    sec  62.2 KBytes  510 Kbits/sec  6.256 ms    0/44 (0%)
[ 5]  8.00-9.00    sec  59.4 KBytes  486 Kbits/sec  6.184 ms    0/42 (0%)
[ 5]  9.00-10.00   sec  62.2 KBytes  510 Kbits/sec  6.309 ms    0/44 (0%)
[ 5] 10.00-11.00   sec  60.8 KBytes  498 Kbits/sec  6.353 ms    0/43 (0%)
[ 5] 11.00-12.00   sec  60.8 KBytes  498 Kbits/sec  6.533 ms    0/43 (0%)
[ 5] 12.00-13.00   sec  60.8 KBytes  498 Kbits/sec  6.398 ms    0/43 (0%)
[ 5] 13.00-14.00   sec  62.2 KBytes  510 Kbits/sec  6.407 ms    0/44 (0%)
[ 5] 14.00-15.00   sec  59.4 KBytes  486 Kbits/sec  6.350 ms    0/42 (0%)
[ 5] 15.00-16.00   sec  62.2 KBytes  510 Kbits/sec  6.143 ms    0/44 (0%)
[ 5] 16.00-17.00   sec  60.8 KBytes  498 Kbits/sec  6.408 ms    0/43 (0%)
[ 5] 17.00-18.00   sec  60.8 KBytes  498 Kbits/sec  6.238 ms    0/43 (0%)
[ 5] 18.00-19.00   sec  62.2 KBytes  510 Kbits/sec  6.435 ms    0/44 (0%)
[ 5] 19.00-20.00   sec  59.4 KBytes  486 Kbits/sec  6.341 ms    0/42 (0%)
[ 5] 20.00-21.00   sec  62.2 KBytes  510 Kbits/sec  6.153 ms    0/44 (0%)
[ 5] 21.00-22.00   sec  60.8 KBytes  498 Kbits/sec  6.421 ms    0/43 (0%)
[ 5] 22.00-23.00   sec  60.8 KBytes  498 Kbits/sec  6.543 ms    0/43 (0%)
[ 5] 23.00-24.00   sec  60.8 KBytes  498 Kbits/sec  6.209 ms    0/43 (0%)
[ 5] 24.00-25.00   sec  60.8 KBytes  498 Kbits/sec  6.039 ms    0/43 (0%)
[ 5] 25.00-26.00   sec  60.8 KBytes  498 Kbits/sec  6.620 ms    0/43 (0%)
[ 5] 26.00-27.00   sec  62.2 KBytes  510 Kbits/sec  6.304 ms    0/44 (0%)
[ 5] 27.00-28.00   sec  60.8 KBytes  498 Kbits/sec  6.310 ms    0/43 (0%)
[ 5] 28.00-29.00   sec  60.8 KBytes  498 Kbits/sec  6.381 ms    0/43 (0%)
[ 5] 29.00-30.00   sec  60.8 KBytes  498 Kbits/sec  6.327 ms    0/43 (0%)
[ 5] 30.00-30.07   sec  4.24 KBytes  473 Kbits/sec  6.331 ms    0/3 (0%)
-----
[ ID] Interval          Transfer      Bitrate      Jitter      Lost/Total Datagrams
[ 5]  0.00-30.07   sec  1.79 MBytes  499 Kbits/sec  6.331 ms    0/1295 (0%) receiver

```

Рисунок 6 – Измерение джиттера

После подробного анализа Cisco IOS можно выделить следующие преимущества:

- высокая стабильность и надёжность работы в реальных сетях;
- хорошая интеграция с другими сетевыми технологиями и сервисами;
- развитые средства мониторинга и диагностики QoS;
- высокая производительность за счёт аппаратной обработки трафика.

Однако данное программное обеспечение имеет и некоторые недостатки. В их числе:

- сложность конфигурации и высокая требовательность к квалификации администратора;
- высокая стоимость оборудования и лицензий;
- зависимость производительности от конкретной аппаратной платформы.

1.2.2 MikroTik RouterOS

MikroTik RouterOS – сетевая операционная система, применяемая в оборудовании MikroTik и ориентированная на использование в малых и средних сетях. Поддерживает базовые и расширенные механизмы QoS, включая очереди, ограничения скорости и приоритизацию трафика. Отличается доступностью и простотой внедрения по сравнению с решениями операторского уровня.

Однако, как и любое программное обеспечение, MikroTik RouterOS имеет свои особенности. Проведем анализ по следующим параметрам:

- простота настройки QoS. Система предоставляет удобный графический интерфейс (WinBox) и CLI, что позволяет быстро настраивать очереди и ограничения трафика. Это делает QoS более доступным для начинающих пользователей, однако при усложнении конфигурации логика политик может становиться менее наглядной;

- гибкость механизмов управления трафиком. RouterOS поддерживает основные инструменты QoS (Simple Queue, Queue Tree, PCQ), чего достаточно для большинства практических задач. Однако по глубине и масштабируемости политик уступает решениям уровня Cisco IOS;

- работа под нагрузкой. Система показывает стабильную производительность в типичных сценариях малых и средних сетей. При высокой нагрузке и большом количестве правил QoS возможно увеличение задержек из-за программной обработки части функций.

В рамках тестирования производительности выполнена загрузка образа MikroTik RouterOS v7 в среду GNS3. При запуске маршрутизатора открывается интерфейс командной строки (CLI), представленный на рисунке 7.



Рисунок 7 – CLI Mikrotik RouterOS v7

Произведена настройка сетевых интерфейсов тестового стенда и подключение маршрутизатора Mikrotik к эмулируемой сети. Конфигурация интерфейсов маршрутизатора приведена на рисунке 8.

```

#  NAME      TYPE      ACTUAL-MTU  MAC-ADDRESS
0  ether1     ether      1500        0C:21:1F:09:00:00
1  ether2     ether      1500        0C:21:1F:09:00:01
2  ether3     ether      1500        0C:21:1F:09:00:02
3  R lo       loopback   65536       00:00:00:00:00:00
[admin@MikroTik] > /interface set ether1 name=LAN1
[admin@MikroTik] > /interface set ether2 name=LAN2
[admin@MikroTik] > /interface set ether3 name=LAN3
[admin@MikroTik] > /interface print
Flags: R - RUNNING
Columns: NAME, TYPE, ACTUAL-MTU, MAC-ADDRESS
#  NAME      TYPE      ACTUAL-MTU  MAC-ADDRESS
0  LAN1      ether      1500        0C:21:1F:09:00:00
1  LAN2      ether      1500        0C:21:1F:09:00:01
2  LAN3      ether      1500        0C:21:1F:09:00:02
3  R lo      loopback   65536       00:00:00:00:00:00
[admin@MikroTik] > /ip address add address=10.0.1.1/24 interface=LAN1
[admin@MikroTik] > /ip address add address=10.0.2.1/24 interface=LAN2
[admin@MikroTik] > /ip address add address=10.0.3.1/24 interface=LAN3

```

Рисунок 8 – Настройка интерфейсов

После завершения конфигурации выполнена генерация сетевой нагрузки между узлами qos-ubuntu-1 и qos-ubuntu-3. Задержка ICMP-пакетов в условиях загруженной сети представлена на рисунке 9.

```

root@qos-ubuntu-1:/# ping 10.0.3.2
PING 10.0.3.2 (10.0.3.2) 56(84) bytes of data.
64 bytes from 10.0.3.2: icmp_seq=3 ttl=63 time=72.5 ms
64 bytes from 10.0.3.2: icmp_seq=4 ttl=63 time=62.7 ms
64 bytes from 10.0.3.2: icmp_seq=6 ttl=63 time=161 ms
64 bytes from 10.0.3.2: icmp_seq=12 ttl=63 time=121 ms
64 bytes from 10.0.3.2: icmp_seq=13 ttl=63 time=172 ms
64 bytes from 10.0.3.2: icmp_seq=19 ttl=63 time=155 ms
64 bytes from 10.0.3.2: icmp_seq=20 ttl=63 time=132 ms
64 bytes from 10.0.3.2: icmp_seq=21 ttl=63 time=168 ms
64 bytes from 10.0.3.2: icmp_seq=24 ttl=63 time=0.303 ms
64 bytes from 10.0.3.2: icmp_seq=25 ttl=63 time=0.334 ms
64 bytes from 10.0.3.2: icmp_seq=26 ttl=63 time=0.335 ms
^C
--- 10.0.3.2 ping statistics ---
26 packets transmitted, 11 received, 57.6923% packet loss, time 26952ms

```

Рисунок 9 - Задержка ICMP пакетов при загруженной сети.

Дополнительно проведено тестирование UDP-трафика между теми же конечными узлами. Целевая скорость генерации трафика составила 500 Кбит/с. Результаты тестирования приведены на рисунке 10.

```

Server listening on 5201
-----
Accepted connection from 10.0.1.10, port 48278
[ 5] local 10.0.2.10 port 5201 connected to 10.0.1.10 port 33529
[ ID] Interval      Transfer      Bitrate      Jitter      Lost/Total Datagrams
[ 5] 0.00-1.00 sec  59.4 KBytes  486 Kbits/sec  0.077 ms    0/42 (0%)
[ 5] 1.00-2.00 sec  60.8 KBytes  498 Kbits/sec  0.073 ms    0/43 (0%)
[ 5] 2.00-3.00 sec  60.8 KBytes  498 Kbits/sec  0.056 ms    0/43 (0%)
[ 5] 3.00-4.00 sec  60.8 KBytes  498 Kbits/sec  0.053 ms    0/43 (0%)
[ 5] 4.00-5.00 sec  60.8 KBytes  498 Kbits/sec  0.048 ms    0/43 (0%)
[ 5] 5.00-6.00 sec  62.2 KBytes  510 Kbits/sec  0.059 ms    0/44 (0%)
[ 5] 6.00-7.00 sec  60.8 KBytes  498 Kbits/sec  0.069 ms    0/43 (0%)
[ 5] 7.00-8.00 sec  60.8 KBytes  498 Kbits/sec  0.033 ms    0/43 (0%)
[ 5] 8.00-9.00 sec  60.8 KBytes  498 Kbits/sec  0.055 ms    0/43 (0%)
[ 5] 9.00-10.00 sec 60.8 KBytes  498 Kbits/sec  0.062 ms    0/43 (0%)
[ 5] 10.00-11.00 sec 60.8 KBytes  498 Kbits/sec  0.055 ms    0/43 (0%)
[ 5] 11.00-12.00 sec 62.2 KBytes  510 Kbits/sec  0.041 ms    0/44 (0%)
[ 5] 12.00-13.00 sec 60.8 KBytes  498 Kbits/sec  0.041 ms    0/43 (0%)
[ 5] 13.00-14.00 sec 60.8 KBytes  498 Kbits/sec  0.052 ms    0/43 (0%)
[ 5] 14.00-15.00 sec 60.8 KBytes  498 Kbits/sec  0.075 ms    0/43 (0%)
[ 5] 15.00-16.00 sec 60.8 KBytes  498 Kbits/sec  0.117 ms    0/43 (0%)
[ 5] 16.00-17.00 sec 60.8 KBytes  498 Kbits/sec  0.063 ms    0/43 (0%)
[ 5] 17.00-18.00 sec 62.2 KBytes  510 Kbits/sec  0.037 ms    0/44 (0%)
[ 5] 18.00-19.00 sec 60.8 KBytes  498 Kbits/sec  0.039 ms    0/43 (0%)
[ 5] 19.00-20.00 sec 60.8 KBytes  498 Kbits/sec  0.047 ms    0/43 (0%)
[ 5] 20.00-21.00 sec 60.8 KBytes  498 Kbits/sec  0.046 ms    0/43 (0%)
[ 5] 21.00-22.00 sec 60.8 KBytes  498 Kbits/sec  0.055 ms    0/43 (0%)
[ 5] 22.00-23.00 sec 60.8 KBytes  498 Kbits/sec  0.031 ms    0/43 (0%)
[ 5] 23.00-24.00 sec 62.2 KBytes  510 Kbits/sec  0.038 ms    0/44 (0%)
[ 5] 24.00-25.00 sec 60.8 KBytes  498 Kbits/sec  0.037 ms    0/43 (0%)
[ 5] 25.00-26.00 sec 60.8 KBytes  498 Kbits/sec  0.037 ms    0/43 (0%)
[ 5] 26.00-27.00 sec 60.8 KBytes  498 Kbits/sec  0.045 ms    0/43 (0%)
[ 5] 27.00-28.00 sec 60.8 KBytes  498 Kbits/sec  0.054 ms    0/43 (0%)
[ 5] 28.00-29.00 sec 60.8 KBytes  498 Kbits/sec  0.049 ms    0/43 (0%)
[ 5] 29.00-30.00 sec 62.2 KBytes  510 Kbits/sec  0.076 ms    0/44 (0%)
[ 5] 30.00-30.04 sec 1.41 KBytes  267 Kbits/sec  0.071 ms    0/1 (0%)
-----
[ ID] Interval      Transfer      Bitrate      Jitter      Lost/Total Datagrams
[ 5] 0.00-30.04 sec 1.79 MBytes  499 Kbits/sec  0.071 ms    0/1295 (0%) receiver

```

Рисунок 10 – Джиттер RouterOS.

1.3 Функциональное сравнение с аналогами

Так как разрабатываемый блок анализа должен функционировать в составе сетевого оборудования, требуется создание программного маршрутизатора, который обеспечит возможность перехвата, анализа и управления сетевым трафиком. Разрабатываемый маршрутизатор имеет ряд сходств с существующими платформами маршрутизации и алгоритмами управления очередями. Улучшение реализации данных возможностей позволит повысить конкурентоспособность проекта относительно распространённых решений в области маршрутизации и управления сетевым трафиком:

- поддержка работы с raw sockets и обработки сетевых пакетов в userspace;
- модульная архитектура с возможностью динамической загрузки алгоритмов планирования через .so библиотеки;
- поддержка многопоточности для раздельной обработки входящего исходящего трафика;
- реализация механизмов управления очередями и ограничения использования памяти;
- возможность внедрения алгоритмов контроля задержек и jitter;
- поддержка Linux-систем без необходимости использования специализированного оборудования.

Однако наличие только базовых функций маршрутизации и очередей недостаточно для конкуренции с промышленными решениями. Для повышения эффективности проекта и расширения его возможностей были предусмотрены следующие архитектурные улучшения:

- поддержка динамической замены алгоритмов планирования пакетов. В отличие от многих готовых решений, система позволяет подключать пользовательские алгоритмы через интерфейс IScheduler без перекомпиляции основного приложения. Это обеспечивает гибкость экспериментирования и исследований;

– реализация собственного механизма контроля памяти очередей. Система отслеживает использование heap memory и предотвращает переполнение памяти при высокой нагрузке;

– минимизация задержек обработки за счёт разделения потоков приёма и передачи трафика. Такая архитектура уменьшает contention между потоками и снижает jitter;

– поддержка исследовательских алгоритмов Active Queue Management (AQM), таких как FIFO, RED и CoDel. Это позволяет анализировать влияние различных дисциплин очередей на задержки, потери пакетов и throughput;

– низкие требования к аппаратным ресурсам. В отличие от аппаратных маршрутизаторов, система может работать на стандартных Linux-серверах и в виртуализированной среде.

Разрабатываемая система также имеет ряд отличий относительно популярных платформ маршрутизации, таких как Cisco Systems IOS и MikroTik RouterOS.

Платформа Cisco IOS ориентирована преимущественно на специализированное сетевое оборудование и корпоративный сегмент. Она обеспечивает высокую стабильность и широкий набор сетевых функций, однако является закрытым коммерческим решением и ограничивает возможность модификации внутренних алгоритмов обработки пакетов.

RouterOS компании MikroTik предоставляет более доступную программно-аппаратную платформу с поддержкой QoS и traffic shaping, однако архитектура системы также остаётся закрытой, а возможности исследования и модификации алгоритмов управления очередями ограничены.

В отличие от данных решений, разрабатываемый маршрутизатор предоставляет:

– открытый исходный код;

– возможность реализации собственных алгоритмов управления очередями;

- гибкую plugin-архитектуру;
- возможность проведения исследований в области QoS и AQM;
- независимость от специализированного оборудования.

Особое внимание в проекте уделяется алгоритмам управления очередями и контроля задержек.

Алгоритм FIFO (First In First Out) представляет собой простейшую дисциплину обслуживания пакетов, при которой пакеты передаются в порядке поступления. Преимуществами FIFO являются простота реализации и минимальные вычислительные затраты, однако при высокой нагрузке возможно возникновение bufferbloat и увеличение задержек.

Алгоритм RED (Random Early Detection) относится к классу Active Queue Management и выполняет вероятностное удаление пакетов до полного заполнения очереди. Это позволяет снизить вероятность перегрузки канала и уменьшить средние задержки передачи данных.

Алгоритм CoDel (Controlled Delay) ориентирован на контроль времени нахождения пакетов в очереди. В отличие от FIFO и RED, CoDel анализирует задержку пакетов и автоматически удаляет их при превышении допустимого уровня latency. Такой подход позволяет эффективно бороться с bufferbloat и снижать jitter при высокой сетевой нагрузке.

Алгоритм ARIMA-pacing (AutoRegressive Integrated Moving Average) представляет собой адаптивный метод управления очередью, основанный на прогнозировании времени прибытия пакетов с помощью статистической модели временных рядов. В отличие от FIFO, RED и CoDel, которые реагируют на текущее состояние очереди, ARIMA-pacing работает на опережение: модель ARIMA(2,1,1) обучается на интервалах между поступлением пакетов и предсказывает момент прихода следующего пакета. На основе прогноза корректируется интервал выпуска пакетов (pacing), что позволяет сглаживать выходной поток и минимизировать джиттер.

Поддержка нескольких алгоритмов в рамках единого интерфейса IScheduler позволяет использовать разработанный маршрутизатор как экспериментальную платформу для исследования сетевых дисциплин обслуживания и методов управления качеством обслуживания трафика (QoS).

1.4 Постановка задачи

Целью выпускной квалификационной работы является разработка и реализация блока анализа темпорально-корреляционных свойств сетевого трафика. Данный блок должен функционировать в составе разрабатываемого программного маршрутизатора. Разрабатываемое программное обеспечение предназначено для анализа и управления транзитным сетевым трафиком с целью снижения сетевых задержек и, в частности, уменьшения величины джиттера. Повышение качества обслуживания (QoS) предполагается достигать за счёт применения механизмов предиктивного выделения ресурсов на основе анализа характеристик и поведения сетевого трафика.

1.5 Выбор инструментов разработки

Комплексный подход к разработке предполагает обоснованный выбор инструментов, используемых в процессе реализации проекта. Предпочтение отдаётся решениям, которые уже освоены либо могут быть быстро изучены, при этом обеспечивая высокую эффективность.

1.5.1 Язык программирования

Для реализации модели анализа сетевого трафика и снижения джиттера могут быть использованы различные языки программирования, среди которых рассмотрены Python, C++ и Go.

C++ является языком системного уровня и обеспечивает высокую производительность, что особенно важно при обработке сетевого трафика в реальном времени. Он предоставляет прямой доступ к низкоуровневым механизмам операционной системы, что позволяет эффективно взаимодействовать с сетевым стеком Linux, включая работу с NFQ-очередями (Netfilter Queue). Кроме того, C++ даёт возможность точно управлять памятью и

реализовывать высокоэффективные структуры данных, такие как FIFO-буферы, используемые для сглаживания задержек. Поддержка многопоточности и высокая скорость выполнения делают его оптимальным выбором для задач, чувствительных к задержкам и джиттеру. Основным недостатком является высокая сложность разработки, необходимость тщательного управления ресурсами и большее время на отладку.

Python характеризуется высокой скоростью разработки и наличием развитой экосистемы библиотек для анализа данных и временных рядов, включая NumPy, SciPy и statsmodels. Это делает его удобным инструментом для прототипирования алгоритмов, реализации моделей прогнозирования (ARIMA/FARIMA), а также проведения имитационного моделирования сетевого трафика. Python позволяет быстро проверять гипотезы и адаптировать алгоритмы без значительных временных затрат. Однако данный язык уступает по производительности и не подходит для обработки сетевых пакетов в реальном времени или взаимодействия с низкоуровневыми механизмами ядра без использования дополнительных расширений.

Go представляет собой современный компилируемый язык, ориентированный на разработку сетевых и распределённых приложений. Его ключевым преимуществом является встроенная поддержка конкурентности через goroutines, что упрощает обработку большого количества сетевых соединений. Go проще в освоении и разработке по сравнению с C++, а также обеспечивает достаточно высокую производительность. Тем не менее, он уступает C++ в задачах, требующих низкоуровневого контроля и максимальной оптимизации, а также имеет ограниченные возможности для реализации сложных математических моделей и анализа временных рядов.

В рамках данной работы целесообразно использовать C++ в качестве основного языка разработки, поскольку он обеспечивает высокую производительность и необходимый уровень контроля над системными ресурсами при работе с сетевым стеком операционной системы. Это особенно

важно при взаимодействии с механизмами ядра Linux, такими как NFQ-очереди, а также при реализации FIFO-буферов для обработки пакетов с минимальными задержками. Python может быть использован как вспомогательный инструмент для моделирования, анализа данных и отработки алгоритмов прогнозирования (например, ARIMA/FARIMA), однако в итоговой системе его применение ограничено задачами, не критичными к производительности.

1.5.2 Среда разработки

Выбор среды разработки является важным этапом, наряду с выбором языка программирования. В качестве целевой операционной системы был выбран Linux, поэтому рассматривались решения, доступные для данной платформы:

– Visual Studio Code. Относится скорее к расширяемым текстовым редакторам, однако за счёт большого количества дополнений может быть преобразован в полноценную среду разработки. Обладает удобным и настраиваемым интерфейсом, поддерживает широкий спектр языков программирования и инструментов. Быстро настраивается под конкретные задачи и предоставляет возможность удалённой разработки через SSH. Существенным недостатком является необходимость периодического доступа к интернету для установки и обновления расширений;

– Apache NetBeans. Полноценная среда разработки с поддержкой нескольких языков программирования, включая Java, C/C++ и PHP. Обладает встроенными инструментами сборки и отладки, что упрощает начальную настройку. Однако интерфейс может показаться перегруженным, а производительность — ниже по сравнению с более лёгкими решениями, особенно на слабых системах;

– Code::Blocks. Среда разработки, ориентированная преимущественно на языки C и C++. Поставляется с базовыми инструментами компиляции и не требует сложной настройки для начала работы. Отличается простотой, но уступает современным решениям по удобству интерфейса и возможностям расширения;

– Kate. Лёгкий текстовый редактор для Linux с поддержкой подсветки синтаксиса и базовых функций разработки. Подходит для простых задач и быстрого редактирования кода, но не предоставляет полноценного набора инструментов IDE, таких как отладка и управление проектами.

Повторно проанализировав требования, описанные ранее, выбор был сделан в пользу Visual Studio Code как наиболее универсального и удобного инструмента для разработки в выбранной среде.

1.5.3 Среда моделирования сетей.

Для разработки и исследования алгоритмов анализа сетевого трафика, а также оценки их влияния на такие параметры, как задержка и джиттер, требуется использование специализированных средств моделирования и эмуляции сетевых инфраструктур. Такие инструменты позволяют воспроизводить поведение реальных сетей, проводить эксперименты в контролируемых условиях и анализировать результаты без необходимости использования физического оборудования.

В рамках данной работы рассматриваются следующие популярные решения: GNS3, Cisco Packet Tracer, EVE-NG.

GNS3 представляет собой мощную среду эмуляции сетей, позволяющую использовать реальные образы операционных систем сетевых устройств. В отличие от классических симуляторов, GNS3 обеспечивает высокий уровень реалистичности за счёт запуска виртуальных маршрутизаторов, коммутаторов и серверов. Поддерживаются реальные образы Cisco IOS, Linux и др. Позволяет производить эмуляцию на виртуальной машине или удаленных серверах и кластерах. Также есть возможность запускать узлы, основанные на docker-контейнерах. Особое значение для данной работы имеет возможность запуска Linux-узлов и взаимодействия с низкоуровневыми механизмами, включая Netfilter и NFQ-очереди. Это позволяет реализовать и протестировать алгоритмы обработки пакетов в условиях, приближенных к реальным. Недостатками GNS3

являются относительно высокая сложность настройки и значительные требования к вычислительным ресурсам.

Cisco Packet Tracer представляет собой учебную среду моделирования сетей, разработанную компанией Cisco. В отличие от систем эмуляции, Packet Tracer использует упрощённые программные модели сетевых устройств вместо реальных образов операционных систем. Это обеспечивает простоту использования и низкие требования к ресурсам, что делает его удобным инструментом для обучения и базового проектирования сетей. Среда поддерживает моделирование маршрутизаторов, коммутаторов и конечных устройств, а также позволяет настраивать основные сетевые протоколы. Имеется встроенная визуализация прохождения пакетов, что упрощает анализ работы сети на базовом уровне. Однако Packet Tracer не выполняет реальное исполнение сетевых операционных систем и не предоставляет доступа к полноценному сетевому стеку. Отсутствует возможность работы с операционной системой Linux на уровне ядра. Это делает невозможной реализацию и тестирование алгоритмов обработки пакетов, ориентированных на низкоуровневое взаимодействие с сетью. Таким образом, несмотря на удобство и доступность, Packet Tracer имеет ограниченную применимость для задач, требующих высокой точности моделирования и анализа временных характеристик трафика.

EVE-NG представляет собой платформу эмуляции сетей, ориентированную на профессиональное использование и построение сложных виртуальных лабораторий. Как и GNS3, данная система поддерживает использование реальных образов сетевых операционных систем, включая Cisco IOS, Linux и другие. Это обеспечивает высокий уровень реалистичности и позволяет моделировать поведение сетей, приближенное к реальным условиям эксплуатации. EVE-NG реализует централизованный подход к управлению через веб-интерфейс и поддерживает развёртывание на серверной инфраструктуре, включая работу в виртуализированных средах и кластерах. Это делает её удобной для масштабируемых сценариев и совместной работы. Платформа также позволяет

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		28

запускать Linux-узлы, что теоретически даёт возможность использования механизмов сетевого стека, включая Netfilter. Однако интеграция пользовательских решений и низкоуровневых механизмов обработки пакетов является менее гибкой по сравнению с GNS3, особенно в контексте прямой разработки и отладки приложений, использующих NFQ-очереди. К недостаткам EVE-NG можно отнести высокие требования к вычислительным ресурсам, а также ограничения функциональности в бесплатной версии. Кроме того, по сравнению с GNS3 система предоставляет меньше возможностей для глубокой кастомизации и интеграции пользовательских инструментов, что может быть критично при разработке специализированных решений.

1.6 Настройка инструментов разработки

В рамках установки рабочего окружения был установлен Debian 11 “Bullseye”. При помощи snap был установлен Visual Studio Code. Установка VS Code приведена на рисунке 11.

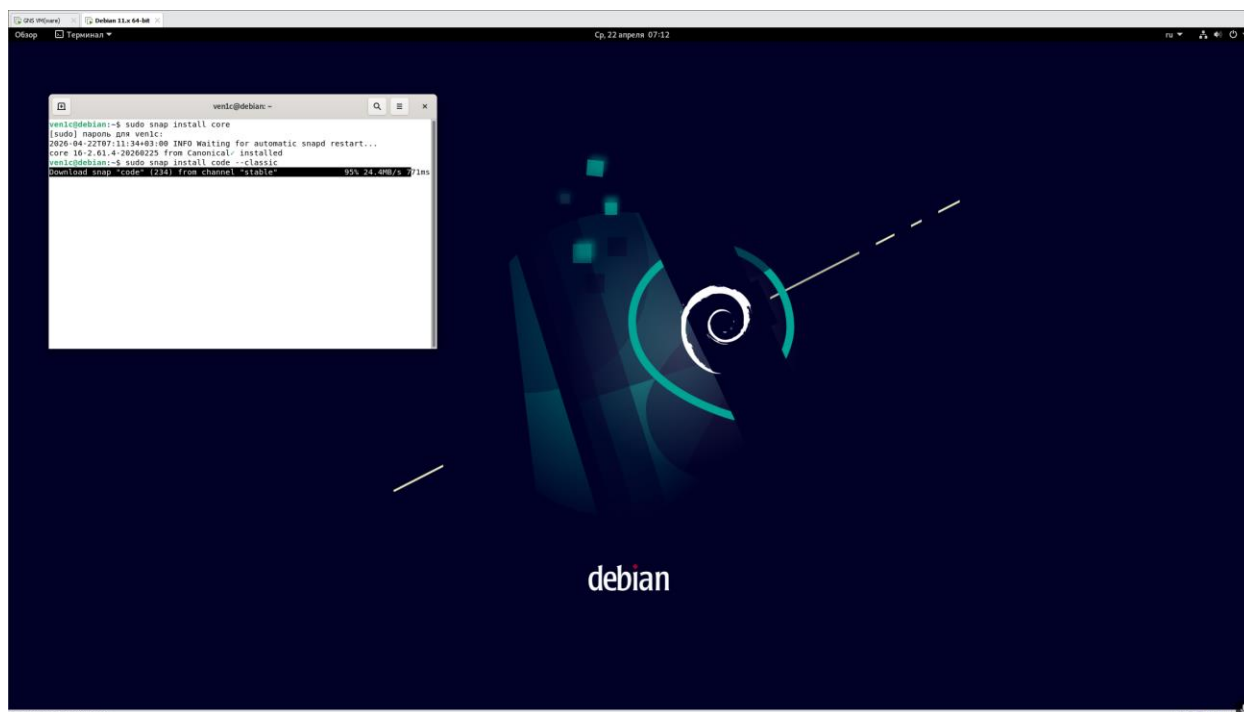


Рисунок 11 – Установка VS Code

					ПГУ1.090301.31.001 ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

В внутреннем менеджере пакетов были установлены пакеты которые потребуются при разработке блока: C/C++, CMake, Makefile. Установка соответствующих пакетов приведена на рисунке 12.

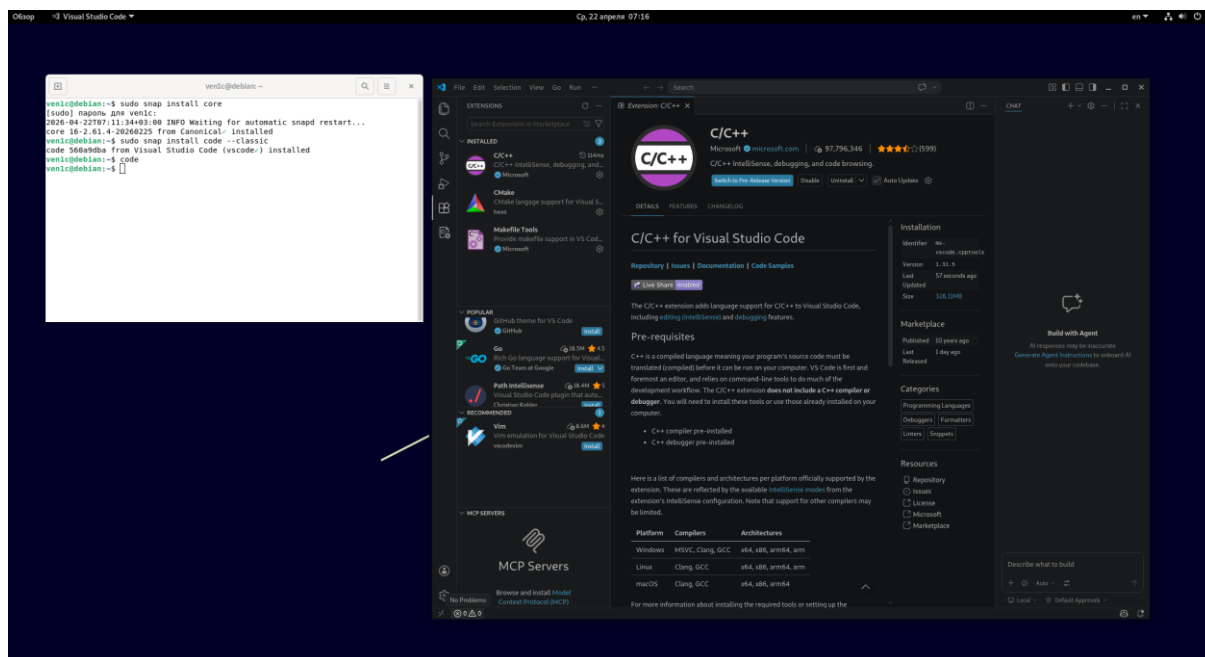


Рисунок 12 – Установка расширений в VS Code

Но так как VS Code не предоставляет собственных инструментов сборки требовалось установить внешний компилятор. Выбор был сделан в пользу gcc и его компонента g++. Установленные пакеты приведены на рисунке 13.

Установка среды GNS3.

При установке GNS3 есть различные варианты работы. В том числе выбор устройств/а на которых будет происходить симуляция компонентов виртуальной сети. Из доступных вариантов:

- удаленный компьютер/кластер;
- локальный компьютер;
- GNS VM.

```

ven1c@debian:~$ gcc --version
g++ --version
make --version
cmake --version
gdb --version
git --version
gcc (Debian 10.2.1-6) 10.2.1 20210110
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

g++ (Debian 10.2.1-6) 10.2.1 20210110
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

GNU Make 4.3
Эта программа собрана для x86_64-pc-linux-gnu
Copyright (C) 1988-2020 Free Software Foundation, Inc.
Лицензия GPLv3+: GNU GPL версии 3 или новее <http://gnu.org/licenses/gpl.html>
Это свободное программное обеспечение: вы можете свободно изменять его и
распространять. НЕТ НИКАКИХ ГАРАНТИЙ вне пределов, допустимых законом.
cmake version 3.18.4

CMake suite maintained and supported by Kitware (kitware.com/cmake).
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
git version 2.30.2
ven1c@debian:~$ █

```

Рисунок 13 – Версии установленных пакетов

Я остановился на последнем варианте, так как по информации от разработчика это позволяет максимально эффективно использовать ресурсы компьютера, ввиду использования максимально эффективного метода виртуализации, для Windows это WSL2. Настройки GNS VM представлены на Рисунке 14.

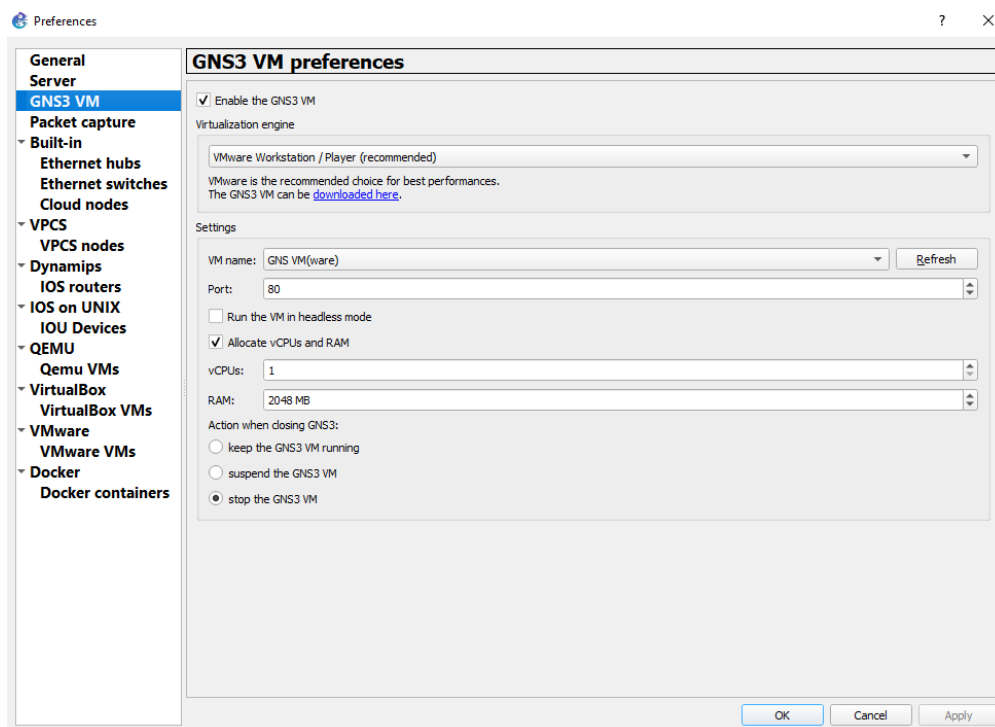


Рисунок 14 – Настройки GNS VM

Так как GNS VM запускается в среде виртуализации VMWare к ней есть полноценный доступ через консоль. После запуска GNS, виртуальная машина автоматически запускается и встречает пользователя приветственным экраном который представлен на рисунке 15.

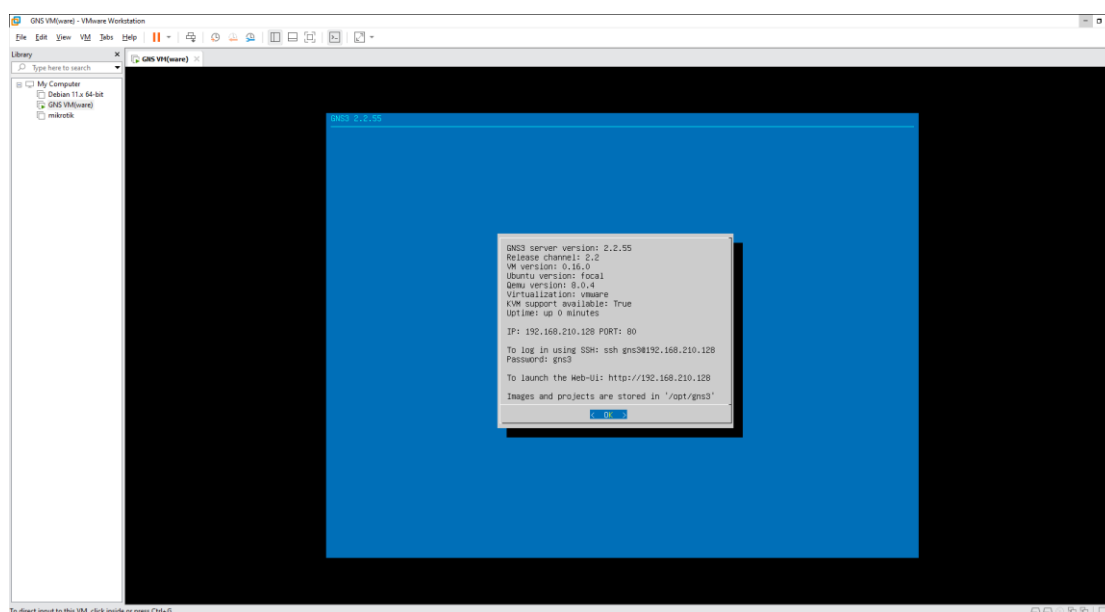


Рисунок 15 – Приветственный экран GNS VM

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.31.001 ПЗ

Лист

32

После данных подготовительных мер я создал две заготовки, для запуска в Docker внутри GNS VM. Первая из них является заготовкой для блока, которая пропускает пакеты внутри себя транзитом(FORWARD) используя очереди NFQ. Вторая же является минимально возможным образом ubuntu в который были установлены пакеты для работы с сетью: ip, iperf, ping и т.д. Созданные контейнеры представлены на Рисунке 16.

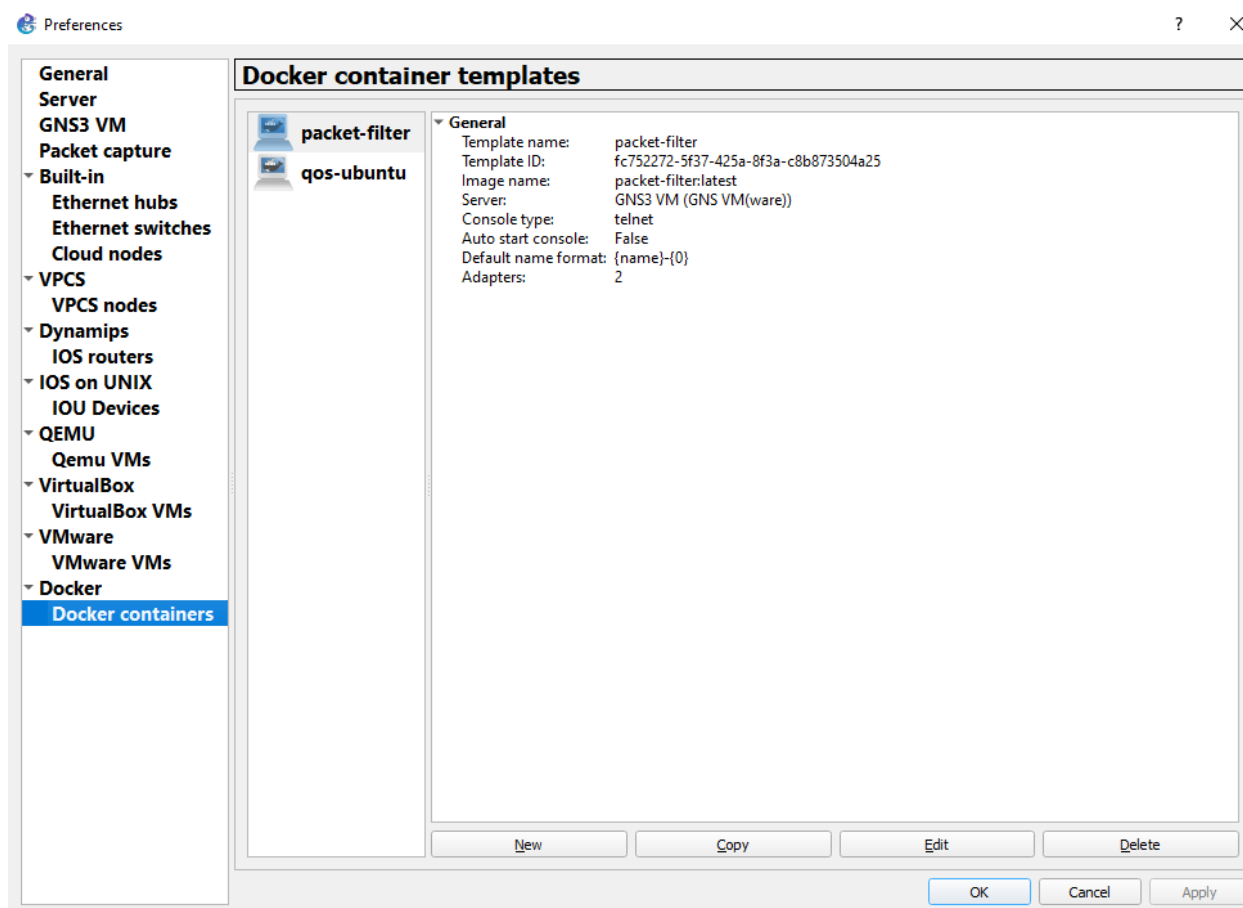


Рисунок 16 – Контейнеры Docker в GNS

1.7 Архитектура разрабатываемой системы

Выбор правильной архитектуры системы является залогом стабильной работы и простоты поддержки в будущем. Для разрабатываемого маршрутизатора, работающего в пространстве пользователя, критически важны производительность обработки пакетов, возможность гибкой замены алгоритмов управления очередью и стабильность при длительной работе. В качестве

возможных вариантов организации вычислительного ядра были рассмотрены однопоточная и многопоточная модели исполнения, а также модульная архитектура с динамической загрузкой компонентов.

Однопоточная архитектура. Однопоточная модель предполагает выполнение всех операций — приём пакетов, их обработку и отправку — в одном потоке команд.

Преимущества однопоточной архитектуры:

- Простота реализации и отладки. Отсутствие состояний гонки (race conditions), взаимных блокировок (deadlocks) и необходимости синхронизации делает код линейным и предсказуемым;

- Детерминированное поведение. Порядок обработки пакетов строго определён, что упрощает анализ очередей и задержек;

- Минимальные накладные расходы. Отсутствуют затраты процессорного времени на переключение контекста и синхронизацию между потоками.

Недостатки однопоточной архитектуры:

- Невозможность параллельной обработки. Пока выполняется операция чтения из одного интерфейса, второй интерфейс простаивает, что может приводить к потере пакетов при высокой нагрузке;

- Блокировка на операциях ввода-вывода. Даже при использовании неблокирующих сокетов, единственный поток вынужден последовательно опрашивать все интерфейсы, что увеличивает задержку;

- Сложность масштабирования. При добавлении новых интерфейсов или усложнении алгоритма очереди производительность падает линейно.

Многопоточная архитектура. Многопоточная модель разделяет задачи приёма, обработки и отправки пакетов по независимым потокам. Для каждого направления (forward и backward) создаются отдельные потоки чтения (RX) и записи (TX), связанные общей очередью-планировщиком. Схема представлена на рисунке 17.

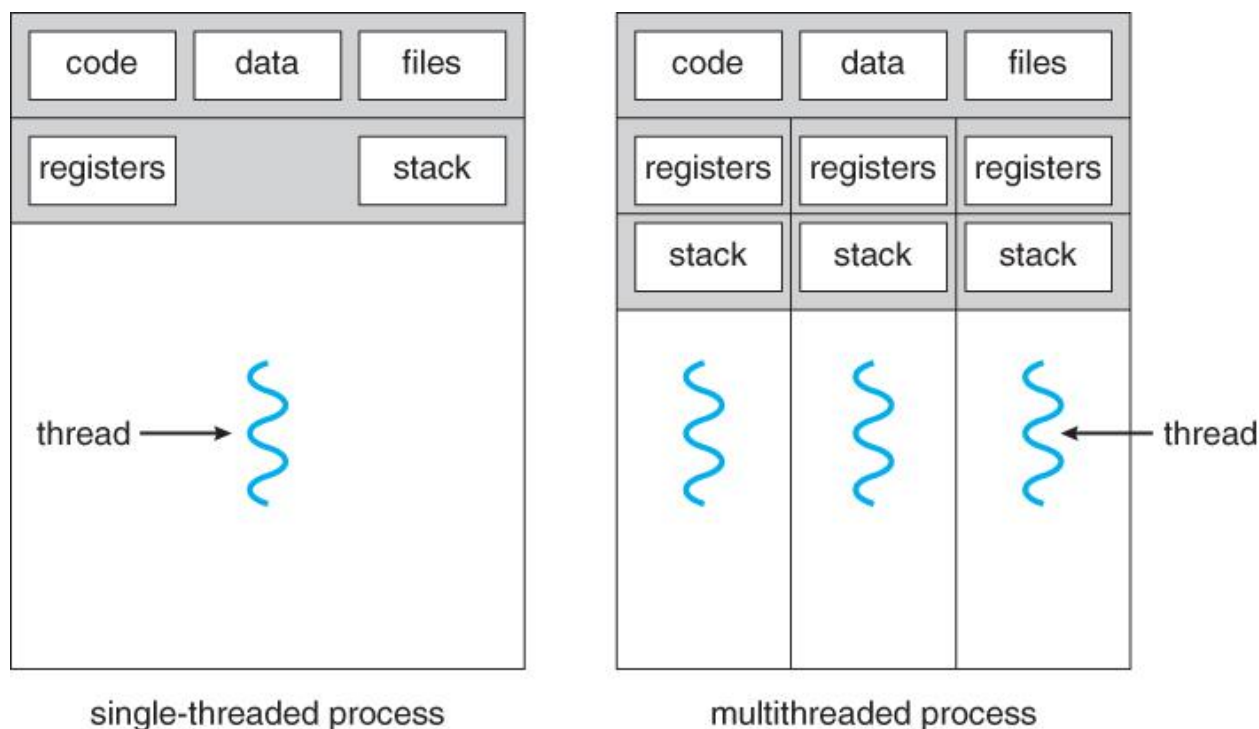


Рисунок 17 – Однопоточная и мультипоточная архитектуры

Преимущества многопоточной архитектуры:

- Параллельная обработка. Потоки чтения и записи работают независимо, что позволяет одновременно принимать пакет на одном интерфейсе и отправлять на другом без взаимных блокировок;
- Разделение ответственности. Каждый поток выполняет строго определённую задачу — либо только чтение из сокета, либо только взаимодействие с планировщиком и отправку;
- Эффективное использование CPU. При простое одного потока (ожидание пакета) другой поток может продолжать обработку очереди;
- Возможность приоритизации. Поток отправки может работать по алгоритму racing с регулируемым интервалом, не влияя на приём пакетов.

Недостатки многопоточной архитектуры:

- Необходимость синхронизации. Общая очередь требует использования мьютексов и условных переменных, что вносит накладные расходы;
- Сложность отладки. Состояния гонки и взаимные блокировки труднее обнаружить и воспроизвести;

– Потребление ресурсов. Каждый поток требует собственный стек и время на переключение контекста планировщиком ОС.

Также при разработке сетевого маршрутизатора принципиальным является выбор уровня операционной системы, на котором будет работать программное обеспечение. Существует два подхода: реализация в пространстве ядра (kernel-space) и реализация в пространстве пользователя (user-space).

Kernel-space реализация. Данный подход предполагает внедрение кода маршрутизатора непосредственно в ядро операционной системы в виде загружаемого модуля (LKM — Loadable Kernel Module) или модификации сетевого стека. Схема linux-kernel представлена на рисунке 18.

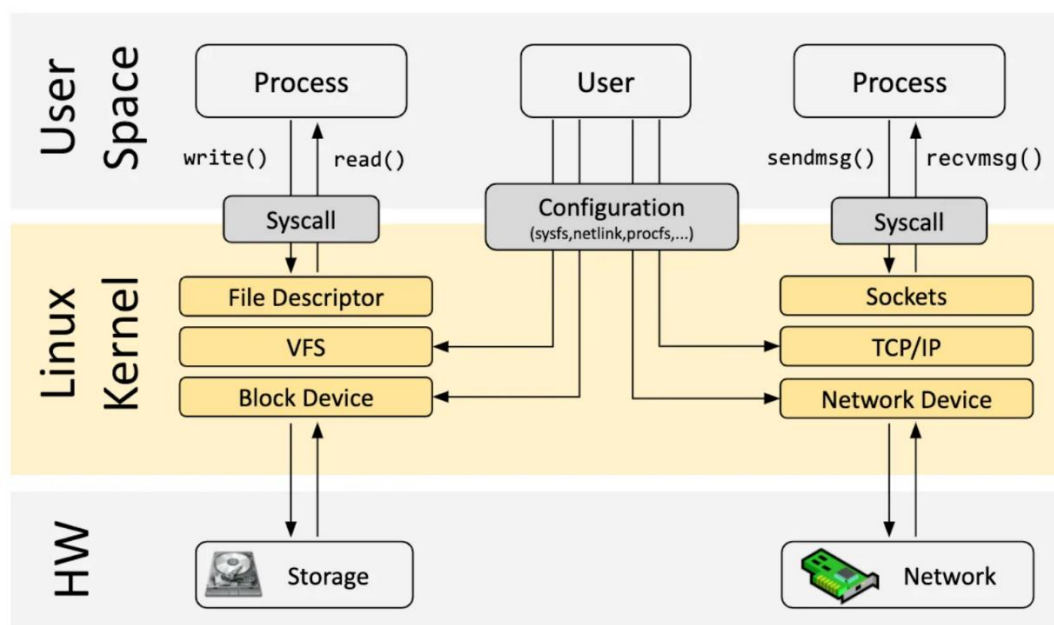


Рисунок 18 – Структура linux-kernel

Преимущества kernel-space реализации:

– Максимальная производительность. Обработка пакетов происходит в контексте ядра без переключения в пользовательское пространство, что исключает накладные расходы на системные вызовы и копирование данных между буферами ядра и пользователя;

– Прямой доступ к сетевым структурам. Модуль ядра имеет непосредственный доступ к `sk_buff`, таблицам маршрутизации и сетевым интерфейсам без промежуточных API;

– Минимальная задержка. Пакет не проходит путь «сетевой интерфейс → ядро → пользователь → ядро → сетевой интерфейс», а обрабатывается полностью внутри ядра;

– Интеграция с Netfilter. Возможность использования стандартных хуков (`NF_INET_PRE_ROUTING`, `NF_INET_POST_ROUTING`) для встраивания в цепочку обработки пакетов.

Недостатки kernel-space реализации:

– Критичность ошибок. Ошибка в модуле ядра (например, разыменование нулевого указателя или утечка памяти) приводит к краху всей системы (`kernel panic`) и требует перезагрузки;

– Сложность разработки и отладки. Отладка кода ядра требует специальных инструментов (`kgdb`, `ftrace`), а тестирование часто невозможно без отдельной виртуальной машины;

– Ограничения языка. Код ядра пишется на C с ограниченным подмножеством стандартной библиотеки (отсутствие `std::vector`, `std::mutex`, исключений C++). Использование C++ в ядре Linux официально не поддерживается;

– Зависимость от версии ядра. API ядра нестабилен: структуры и функции могут меняться между версиями, что требует постоянной адаптации модуля;

User-space реализация. Пользовательский подход использует стандартные API операционной системы — сырые сокеты (`AF_PACKET`, `AF_INET`, `SOCK_RAW`) — для приёма и отправки пакетов из пользовательского процесса.

Преимущества user-space реализации:

– Безопасность и стабильность. Ошибка в процессе роутера приводит только к его завершению, но не затрагивает ядро и другие процессы системы. Возможен автоматический перезапуск через `systemd`;

– Простота разработки. Доступны все возможности C++: STL, исключения, многопоточность (`std::thread`, `std::mutex`), библиотеки линейной алгебры (GSL) для реализации алгоритмов прогнозирования;

– Переносимость. Код использует POSIX-совместимые API и может быть перенесён между различными дистрибутивами Linux и даже на другие UNIX-подобные системы (FreeBSD) с минимальными изменениями;

– Удобство отладки. Доступны стандартные инструменты: `gdb`, `valgrind`, `strace`, `tcpdump`. Логирование реализуется стандартными средствами C++ без ограничений ядерного `printf`;

– Гибкая система плагинов. Динамическая загрузка разделяемых библиотек (`.so`) через `dlopen` позволяет менять алгоритмы управления очередью без пересборки и перезагрузки системы;

– Отсутствие привязки к версии ядра. Пользовательские API (`socket`, `sendto`, `recv`) стабильны десятилетиями.

Недостатки user-space реализации:

– Дополнительные накладные расходы. Каждый пакет проходит путь «ядро → пользователь → ядро», что включает копирование данных и системные вызовы. На скоростях 1-10 Гбит/с это может стать узким местом;

– Повышенная задержка. Переключение контекста между ядром и пользователем добавляет единицы-десятки микросекунд к времени обработки каждого пакета;

– Ограниченная пропускная способность. При очень высоких скоростях (10+ Гбит/с) однопоточная обработка в user-space не справляется без применения технологий обхода ядра (DPDK, XDP, AF_XDP);

– Конкуренция за сокеты. Другие приложения (tcpdump, сетевые менеджеры) могут перехватывать пакеты из AF_PACKET сокета, что требует фильтрации трафика на уровне приложения.

Многопоточная архитектура — выбрана вместо однопоточной ввиду необходимости одновременной обработки двух дуплексных направлений (forward и backward). Разделение на потоки чтения (RX) и записи (TX) для каждого направления позволило исключить взаимные блокировки при операциях ввода-вывода и реализовать алгоритмы pacing с регулируемым интервалом без влияния на скорость приёма пакетов.

Реализация в пространстве пользователя (user-space) — выбрана вместо kernel-space модуля, поскольку проект носит исследовательский характер. User-space подход обеспечил использование современных средств C++ (STL, многопоточность, исключения), библиотеки линейной алгебры GSL для реализации ARIMA-прогнозирования, а также безопасное тестирование экспериментальных алгоритмов без риска краха системы. Целевая нагрузка 50-100 Мбит/с значительно ниже порога, при котором накладные расходы user-space становятся критичными.

Модульная архитектура с динамической загрузкой плагинов — применена для реализации сменных алгоритмов управления очередью (FIFO, RED, CoDel, ARIMA). Использование разделяемых библиотек (.so) с загрузкой через dlopen/dlsym позволило менять планировщик путём редактирования конфигурационного файла без перекомпиляции основного кода роутера, что критично для проведения серии экспериментов в рамках работы.

1.8 Разработка маршрутизатора и модулей очереди

1.8.1 Разработка классов

Разработанный программный маршрутизатор реализован на языке C++17 с использованием объектно-ориентированного подхода. Архитектура системы представлена в виде диаграммы классов на рисунке 19.

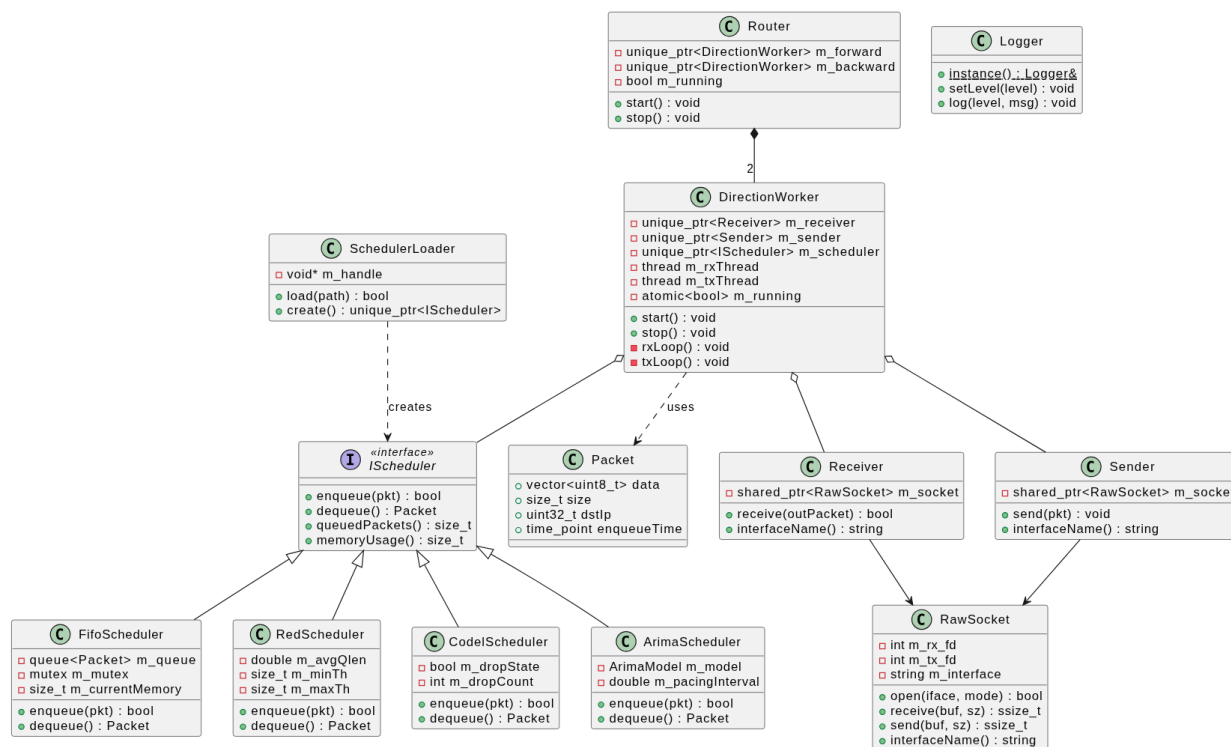


Рисунок 19 - Диаграмма классов

Центральным элементом является класс Router, управляющий двумя экземплярами DirectionWorker для прямого и обратного направлений маршрутизации. Каждый DirectionWorker содержит объекты Receiver, Sender и IScheduler, связанные отношением композиции. Потоки приёма и отправки работают независимо в рамках одного направления, синхронизируясь через общий планировщик очереди.

Взаимодействие с сетевыми интерфейсами инкапсулировано в классе RawSocket, поддерживающем два режима работы: RX и TX. Классы Receiver и Sender используют RawSocket через разделяемое владение (std::shared_ptr).

Интерфейс IScheduler определяет контракт для всех алгоритмов управления очередью. Четыре конкретные реализации — FifoScheduler, RedScheduler, CodelScheduler и ArimaScheduler — наследуют данный интерфейс и загружаются динамически через класс SchedulerLoader.

Вспомогательные классы Logger и ConfigLoader обеспечивают логирование с настраиваемым уровнем детализации и чтение параметров конфигурации из INI-файла соответственно

1.8.2 Класс Packet

Предназначен для инкапсуляции сетевого пакета, передаваемого между компонентами маршрутизатора. Содержит сырые данные IP-пакета, его размер, IP-адрес назначения и временную метку постановки в очередь. Класс реализует семантику перемещения (move semantics) для эффективной передачи владения данными между потоками без избыточного копирования. Конструктор копирования удалён для предотвращения случайного дублирования больших объёмов данных. Временная метка enqueueTime устанавливается автоматически в момент создания объекта и используется алгоритмами управления очередью (CoDel, ARIMA) для измерения времени пребывания пакета в очереди. Использование отдельного класса для представления пакета позволяет унифицировать интерфейс между приёмником, планировщиком и передатчиком.

1.8.3 Класс RawSocket

Инкапсулирует низкоуровневую работу с сетевыми сокетами операционной системы Linux. Поддерживает два режима работы: RX (приём) и TX (передача). В режиме RX используется сокет типа AF_PACKET + SOCK_RAW + ETH_P_ALL с привязкой к конкретному интерфейсу через системный вызов bind(), что обеспечивает захват всех Ethernet-кадров, поступающих на заданный интерфейс. В режиме TX используется сокет типа AF_INET + SOCK_RAW + IPPROTO_RAW с опцией IP_HDRINCL и привязкой к интерфейсу через SO_BINDTODEVICE, что позволяет отправлять самостоятельно сформированные IP-пакеты с полным контролем над заголовком. Оба сокета переводятся в неблокирующий режим (O_NONBLOCK). Класс предоставляет методы receive() и send(), скрывающие детали системных вызовов и обработку ошибок (EAGAIN, EINTR). Отдельный класс для работы с сокетами изолирует платформозависимый код от остальной системы.

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		41

1.8.4 Класс Receiver

Реализует логику приёма сетевых пакетов из RAW-сокета. Принимает в конструкторе разделяемый указатель на объект RawSocket, настроенный в режиме RX. Метод receive() выполняет чтение данных из сокета в буфер, проверяет минимальный размер полученного кадра (Ethernet-заголовок 14 байт + IP-заголовок 20 байт), отсекает Ethernet-заголовок, выделяя чистый IP-пакет, и фильтрует нежелательный трафик: широковещательные пакеты (INADDR_BROADCAST), пакеты с нулевым адресом отправителя и пакеты не IPv4. Отфильтрованный IP-пакет упаковывается в объект Packet и возвращается вызывающему коду. Отделение логики приёма в самостоятельный класс позволяет модифицировать правила фильтрации без изменения остальных компонентов.

1.8.5 Класс Sender

Реализует логику отправки IP-пакетов через RAW-сокеты. Принимает в конструкторе разделяемый указатель на объект RawSocket, настроенный в режиме TX. Метод send() принимает пакет по rvalue-ссылке, извлекает из него IP-заголовок, проверяет время жизни (TTL): если $TTL \leq 1$, пакет отбрасывается. При допустимом TTL значение уменьшается на единицу, после чего пересчитывается контрольная сумма IP-заголовка функцией checksum(). Отправка выполняется через sendto() с указанием адреса назначения, извлечённого из IP-заголовка. Класс инкапсулирует логику модификации заголовка и контрольной суммы, предоставляя остальной системе простой интерфейс отправки.

1.8.6 Класс IScheduler

Абстрактный интерфейс планировщика очереди пакетов. Определяет контракт для всех подключаемых алгоритмов управления очередью. Содержит чисто виртуальные методы: enqueue() — поместить пакет в очередь, dequeue() — извлечь пакет из очереди, queuedPackets() — получить текущее количество пакетов в очереди, memoryUsage() — получить текущий объём занятой памяти. Наследование от этого интерфейса позволяет реализовать различные алгоритмы

(FIFO, RED, CoDel, ARIMA-pacing) в виде взаимозаменяемых модулей, загружаемых динамически.

1.8.7 Класс SchedulerLoader

Обеспечивает динамическую загрузку планировщиков очереди из разделяемых библиотек (.so). Метод load() принимает путь к файлу библиотеки, загружает её через dlopen() и разрешает символ createScheduler через dlsym(). Метод create() вызывает фабричную функцию и возвращает умный указатель на созданный объект планировщика. Класс изолирует механизм динамической загрузки от основного кода роутера, позволяя менять алгоритм очереди путём редактирования конфигурационного файла без перекомпиляции.

1.8.8 Класс DirectionWorker

Реализует обработку одного направления маршрутизации (например, ens36 → ens37). Содержит два потока: поток приёма (RX), который циклически читает пакеты из Receiver и помещает их в IScheduler, и поток отправки (TX), который циклически извлекает пакеты из IScheduler и передаёт их в Sender. Потоки синхронизируются через общий планировщик очереди, использующий мьютекс и условную переменную. Методы start() и stop() управляют жизненным циклом потоков. Флаг m_running типа std::atomic<bool> обеспечивает корректное завершение потоков без гонок данных. Разделение направлений на независимые экземпляры DirectionWorker позволяет обрабатывать дуплексный трафик параллельно.

1.8.9 Класс Router

Композиционный класс, объединяющий два экземпляра DirectionWorker для прямого (forward) и обратного (backward) направлений маршрутизации. Метод start() запускает оба направления, метод stop() останавливает их. Деструктор гарантирует остановку при разрушении объекта. Класс реализует паттерн «Фасад», скрывая внутреннюю многопоточную структуру за простым интерфейсом запуска и остановки.

1.8.10 Класс Logger

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		43

Отвечает за логирование событий внутри маршрутизатора. Реализован как синглтон (паттерн «одиночка») для обеспечения глобального доступа из любого компонента системы. Имеет следующие уровни логирования:

- DEBUG. Отладочные сообщения, интересующие только разработчика. Включают информацию о полученных байтах, состоянии очереди, времени ожидания пакетов;

- INFO. Информационные сообщения. Оповещают о запуске роутера, загрузке плагинов, выбранных интерфейсах;

- WARN. Предупреждения о нештатных, но не критичных ситуациях;

- ERROR. Сообщения об ошибках, не нарушающих общую работу (например, сбой отправки пакета).

Запись лога производится через макросы LOG_DEBUG, LOG_INFO, LOG_WARN, LOG_ERROR. Сообщения выводятся в консоль и, при необходимости, в файл. Каждая запись содержит временную метку с миллисекундами, идентификатор потока (TID) и уровень логирования. Метод setLevel() устанавливает минимальный уровень отображаемых сообщений. Использование отдельного класса для логирования позволяет изменять формат вывода и способ хранения без модификации остальных компонентов.

1.8.11 Класс ConfigLoader

Обеспечивает чтение конфигурации из INI-файла. Метод load() построчно разбирает файл, выделяя секции (в квадратных скобках) и пары «ключ=значение». Поддерживает комментарии (строки, начинающиеся с # или ;). Метод get() возвращает значение параметра по имени секции и ключу с возможностью задания значения по умолчанию. Класс позволяет изменять настройки роутера (интерфейсы, путь к плагину, уровень логирования) без перекомпиляции.

1.8.10 Класс FifoScheduler

Реализует классическую очередь First-In-First-Out с отбрасыванием хвоста (tail drop) при переполнении. При превышении лимита памяти (128 МБ) новый

пакет отбрасывается. Использует `std::queue`, мьютекс и условную переменную для потокобезопасного доступа. Наследует `IScheduler`.

1.8.11 Класс `RedScheduler`

Реализует алгоритм `Random Early Detection (RED)` — упреждающего случайного отбрасывания пакетов. Вычисляет экспоненциально взвешенное скользящее среднее длины очереди. При превышении минимального порога начинает случайное отбрасывание с линейно растущей вероятностью, при превышении максимального порога отбрасывает все пакеты. Параметры: минимальный порог — 100 пакетов, максимальный — 300 пакетов, максимальная вероятность отбрасывания — 10%, вес скользящего среднего — 0,002. Наследует `IScheduler`.

1.8.12 Класс `CodelScheduler`

Реализует алгоритм `Controlled Delay (CoDel)` для борьбы с `bufferbloat`. Отслеживает время пребывания пакета в очереди (`sojourn time`). Если задержка превышает целевое значение (5 мс) в течение интервала (100 мс), переходит в режим отбрасывания с увеличивающейся частотой, определяемой формулой $\text{interval} / \sqrt{(\text{drop_count})}$. При падении задержки ниже цели выходит из режима отбрасывания. Наследует `IScheduler`.

1.8.13 Класс `ArimaScheduler`

Реализует алгоритм `ricing` с прогнозированием на основе модели `ARIMA(2,1,1)`. Содержит вложенный класс `ArimaModel`, выполняющий оценку коэффициентов авторегрессии методом Юла-Уокера через LU-разложение матрицы Тейлора с использованием библиотеки `GSL`. Пересчёт модели производится раз в 10 пакетов для минимизации накладных расходов. Память под матрицы выделяется однократно в конструкторе. Выпуск пакетов производится с адаптивным интервалом (`ricing`), который корректируется по прогнозу `ARIMA` и ошибке между плановым и фактическим интервалом выпуска. `Ricing` применяется только при длине очереди не более трёх пакетов,

чтобы не создавать избыточную задержку при росте нагрузки. Наследует IScheduler.

1.9 Тестирование и отладка

Тестирование разработанного маршрутизатора проводилось в среде эмуляции сетей GNS3 с использованием утилиты генерации трафика iperf3. Для оценки эффективности алгоритмов управления очередью выбран режим UDP с измерением джиттера на скорости 50 Мбит/с в течение 30 секунд. Запуск теста осуществляется данными командами:

- iperf3 -s
- iperf3 -c 10.0.2.10 -u -b 50M -t 30

Схема подключения тестового стенда показана на рисунке X. Стенд состоит из трёх узлов: маршрутизатора под управлением разработанного программного обеспечения и двух конечных узлов, выступающих в роли отправителя и получателя трафика. Маршрутизатор подключён к сетям Vmnet2 и Vmnet3 через интерфейсы ens36 и ens37 соответственно. Конечные узлы находятся в соответствующих сетях и обмениваются трафиком исключительно через маршрутизатор. Запуск маршрутизатора на выделенной виртуальной машине выполняется следующей последовательностью команд, обеспечивающей настройку сетевых интерфейсов и подавление нежелательных ICMP-сообщений ядра:

- sudo ip link set ens36 up
- sudo ip link set ens37 up
- sudo ip addr add 10.0.1.1/24 dev ens36
- sudo ip addr add 10.0.2.1/24 dev ens37
- sudo iptables -A OUTPUT -p icmp --icmp-type destination-unreachable -j

DROP

- cd ~/project_router/build
- sudo ./router

Назначение IP-адресов на интерфейсы маршрутизатора необходимо для обработки ARP-запросов от конечных узлов, так как без них виртуальные машины не смогут определить MAC-адрес шлюза. Правило iptables блокирует генерацию ядром сообщений «Destination Host Unreachable» в ответ на пакеты, адресованные в смежную подсеть — эти сообщения являются паразитным трафиком, искажающим результаты измерений.

Схема подключения стенда показана на рисунке 20.



Рисунок 20 – Схема подключения стенда

Конечные узлы qos-ubuntu-1 и qos-ubuntu-3 реализованы в виде Docker-контейнеров, развёрнутых внутри виртуальной машины GNS3. Использование контейнеризации позволило унифицировать окружение и упростить развёртывание тестового стенда. Dockerfile данных контейнеров приведён на рисунке 21. Сборка образа производилась следующей командой:

`docker build -t qos-ubuntu .`

```
GNU nano 4.8
FROM ubuntu:22.04

RUN apt update && apt install -y \
    iproute2 \
    iputils-ping \
    iperf3 \
    net-tools \
    tcpdump \
    curl \
    && rm -rf /var/lib/apt/lists/*

CMD ["/bin/bash"]
```

Рисунок 21 – Схема подключения стенда

Сетевые настройки конечных узлов тестового стенда приведены в таблице 2. Настройка адресации и маршрутизации на каждом узле выполнена командами:

- ip addr add <IP-адрес> dev eth0
- ip route add default via <шлюз> dev eth0

Таблица 2 – Сравнение аналогов по критериям

Узел	Интерфейс	IP-адрес	Шлюз по умолчанию	Сеть
Маршрутизатор	ens36	10.0.1.1/24	-	10.0.1.0/24
Маршрутизатор	ens37	10.0.2.1/24	-	10.0.2.0/24
qos-ubuntu-1	eth0	10.0.1.10/24	10.0.1.1	10.0.1.0/24
qos-ubuntu-3	eth0	10.0.2.10/24	10.0.2.1	10.0.2.0/24

Маршрут по умолчанию на каждом узле направлен на IP-адрес соответствующего интерфейса маршрутизатора, что обеспечивает прохождение всего межузлового трафика через разработанную систему. Тестирование включает поочередную работу маршрутизатора с каждым из реализованных планировщиков очереди (FIFO, RED, CoDel, ARIMA-pacing) и сбор статистики по пропускной способности, задержке и джиттеру.

1.9.1 Тестирование FIFO

Тестирование алгоритма FIFO проводилось в течение 30 секунд с генерацией UDP-трафика на скорости 50 Мбит/с. Результаты измерений представлены на рисунке 22.

Анализ полученных результатов показывает, что алгоритм FIFO при заданной нагрузке обеспечивает полную доставку пакетов без потерь. Пропускная способность составила 49,9 Мбит/с, что практически соответствует номинальной скорости генерации трафика (50 Мбит/с). Незначительное

расхождение объясняется накладными расходами на инкапсуляцию и передачу служебных заголовков.

Джиттер на уровне 0,427 мс является приемлемым для большинства сетевых приложений, однако при увеличении нагрузки следует ожидать роста данного показателя, поскольку алгоритм FIFO не содержит механизмов сглаживания выходного потока. Отсутствие потерянных пакетов (0 из 129 487) свидетельствует о достаточности буферной памяти (128 МБ) для заданной интенсивности трафика — очередь ни разу не достигала порога переполнения за время эксперимента.

```

-----
Server listening on 5201
-----
Accepted connection from 10.0.1.10, port 34610
[ 5] local 10.0.2.10 port 5201 connected to 10.0.1.10 port 48183
[ ID] Interval          Transfer      Bitrate      Jitter      Lost/Total Datag
rams
[ 5] 0.00-1.00 sec 5.70 MBytes 47.8 Mbits/sec 0.345 ms 0/4129 (0%)
[ 5] 1.00-2.00 sec 5.96 MBytes 50.0 Mbits/sec 0.161 ms 0/4315 (0%)
[ 5] 2.00-3.00 sec 5.96 MBytes 50.0 Mbits/sec 0.408 ms 0/4317 (0%)
[ 5] 3.00-4.00 sec 5.96 MBytes 50.0 Mbits/sec 0.154 ms 0/4316 (0%)
[ 5] 4.00-5.00 sec 5.96 MBytes 50.0 Mbits/sec 0.301 ms 0/4314 (0%)
[ 5] 5.00-6.00 sec 5.96 MBytes 50.0 Mbits/sec 0.039 ms 0/4319 (0%)
[ 5] 6.00-7.00 sec 5.96 MBytes 50.0 Mbits/sec 0.370 ms 0/4314 (0%)
[ 5] 7.00-8.00 sec 5.96 MBytes 50.0 Mbits/sec 0.055 ms 0/4317 (0%)
[ 5] 8.00-9.00 sec 5.96 MBytes 50.0 Mbits/sec 0.385 ms 0/4316 (0%)
[ 5] 9.00-10.00 sec 5.96 MBytes 50.0 Mbits/sec 0.178 ms 0/4318 (0%)
[ 5] 10.00-11.00 sec 5.96 MBytes 50.0 Mbits/sec 0.130 ms 0/4316 (0%)
[ 5] 11.00-12.00 sec 5.96 MBytes 50.0 Mbits/sec 0.377 ms 0/4316 (0%)
[ 5] 12.00-13.00 sec 5.96 MBytes 50.0 Mbits/sec 0.235 ms 0/4315 (0%)
[ 5] 13.00-14.00 sec 5.96 MBytes 50.0 Mbits/sec 0.075 ms 0/4317 (0%)
[ 5] 14.00-15.00 sec 5.95 MBytes 50.0 Mbits/sec 0.118 ms 0/4312 (0%)
[ 5] 15.00-16.00 sec 5.97 MBytes 50.0 Mbits/sec 0.257 ms 0/4321 (0%)
[ 5] 16.00-17.00 sec 5.96 MBytes 50.0 Mbits/sec 0.234 ms 0/4317 (0%)
[ 5] 17.00-18.00 sec 5.96 MBytes 50.0 Mbits/sec 0.130 ms 0/4314 (0%)
[ 5] 18.00-19.00 sec 5.96 MBytes 50.0 Mbits/sec 0.239 ms 0/4319 (0%)
[ 5] 19.00-20.00 sec 5.96 MBytes 50.0 Mbits/sec 0.342 ms 0/4315 (0%)
[ 5] 20.00-21.00 sec 5.96 MBytes 50.0 Mbits/sec 0.393 ms 0/4317 (0%)
[ 5] 21.00-22.00 sec 5.96 MBytes 50.0 Mbits/sec 0.283 ms 0/4315 (0%)
[ 5] 22.00-23.00 sec 5.96 MBytes 50.0 Mbits/sec 0.163 ms 0/4317 (0%)
[ 5] 23.00-24.00 sec 5.96 MBytes 50.0 Mbits/sec 0.354 ms 0/4316 (0%)
[ 5] 24.00-25.00 sec 5.96 MBytes 50.0 Mbits/sec 0.161 ms 0/4316 (0%)
[ 5] 25.00-26.00 sec 5.96 MBytes 50.0 Mbits/sec 0.063 ms 0/4318 (0%)
[ 5] 26.00-27.00 sec 5.96 MBytes 50.0 Mbits/sec 0.359 ms 0/4314 (0%)
[ 5] 27.00-28.00 sec 5.96 MBytes 50.0 Mbits/sec 0.401 ms 0/4317 (0%)
[ 5] 28.00-29.00 sec 5.96 MBytes 50.0 Mbits/sec 0.380 ms 0/4316 (0%)
[ 5] 29.00-30.00 sec 5.96 MBytes 50.0 Mbits/sec 0.161 ms 0/4318 (0%)
[ 5] 30.00-30.04 sec 263 KBytes 49.9 Mbits/sec 0.427 ms 0/186 (0%)
-----
[ ID] Interval          Transfer      Bitrate      Jitter      Lost/Total Datag
rams
[ 5] 0.00-30.04 sec 179 MBytes 49.9 Mbits/sec 0.427 ms 0/129487 (0%) r
eceiver
-----

```

Рисунок 22 – Джиттер FIFO

Таким образом, алгоритм FIFO демонстрирует стабильную работу при умеренной нагрузке, однако простота его логики не позволяет гарантировать сохранение низкого джиттера и нулевых потерь при пиковых нагрузках или изменении характера трафика, что обосновывает необходимость применения более сложных алгоритмов активного управления очередью.

1.9.2 Тестирование RED

Тестирование алгоритма RED (Random Early Detection) проводилось в идентичных условиях: длительность 30 секунд, UDP-трафик с номинальной скоростью 50 Мбит/с. Параметры алгоритма установлены по умолчанию: минимальный порог очереди — 100 пакетов, максимальный порог — 300 пакетов, максимальная вероятность отбрасывания — 10%. Результаты измерений представлены на рисунке 23

```

Server listening on 5201
-----
Accepted connection from 10.0.1.10, port 57164
[ 5] local 10.0.2.10 port 5201 connected to 10.0.1.10 port 46486
[ ID] Interval      Transfer    Bitrate      Jitter      Lost/Total Datagrams
[ 5]  0.00-1.00    sec  4.92 MBytes  41.2 Mbits/sec  0.458 ms    209/3769 (5.5%)
[ 5]  1.00-2.00    sec  4.71 MBytes  39.5 Mbits/sec  0.341 ms    874/4286 (20%)
[ 5]  2.00-3.00    sec  5.42 MBytes  45.5 Mbits/sec  0.359 ms    474/4399 (11%)
[ 5]  3.00-4.00    sec  5.57 MBytes  46.7 Mbits/sec  0.575 ms    252/4286 (5.9%)
[ 5]  4.00-5.00    sec  5.97 MBytes  50.1 Mbits/sec  0.300 ms    140/4461 (3.1%)
[ 5]  5.00-6.00    sec  5.38 MBytes  45.1 Mbits/sec  0.634 ms    237/4132 (5.7%)
[ 5]  6.00-7.00    sec  6.08 MBytes  51.0 Mbits/sec  0.312 ms    240/4644 (5.2%)
[ 5]  7.00-8.00    sec  4.67 MBytes  39.2 Mbits/sec  0.324 ms    591/3973 (15%)
[ 5]  8.00-9.00    sec  5.39 MBytes  45.2 Mbits/sec  0.317 ms    536/4437 (12%)
[ 5]  9.00-10.00   sec  5.55 MBytes  46.5 Mbits/sec  0.363 ms    348/4366 (8%)
[ 5] 10.00-11.00   sec  5.48 MBytes  46.0 Mbits/sec  0.378 ms    307/4275 (7.2%)
[ 5] 11.00-12.00   sec  5.37 MBytes  45.0 Mbits/sec  0.494 ms    369/4256 (8.7%)
[ 5] 12.00-13.00   sec  5.28 MBytes  44.3 Mbits/sec  0.574 ms    541/4364 (12%)
[ 5] 13.00-14.00   sec  5.74 MBytes  48.2 Mbits/sec  0.515 ms    352/4512 (7.8%)
[ 5] 14.00-15.00   sec  5.68 MBytes  47.6 Mbits/sec  0.308 ms    88/4202 (2.1%)
[ 5] 15.00-16.00   sec  5.72 MBytes  48.0 Mbits/sec  0.298 ms    173/4315 (4%)
[ 5] 16.00-17.00   sec  5.67 MBytes  47.5 Mbits/sec  0.397 ms    174/4279 (4.1%)
[ 5] 17.00-18.00   sec  5.62 MBytes  47.1 Mbits/sec  0.282 ms    172/4240 (4.1%)
[ 5] 18.00-19.00   sec  5.33 MBytes  44.8 Mbits/sec  0.499 ms    428/4291 (10%)
[ 5] 19.00-20.00   sec  5.30 MBytes  44.5 Mbits/sec  0.314 ms    629/4467 (14%)
[ 5] 20.00-21.00   sec  5.70 MBytes  47.8 Mbits/sec  0.218 ms    368/4498 (8.2%)
[ 5] 21.00-22.00   sec  5.96 MBytes  50.0 Mbits/sec  0.342 ms     0/4317 (0%)
[ 5] 22.00-23.00   sec  5.96 MBytes  50.0 Mbits/sec  0.319 ms     0/4317 (0%)
[ 5] 23.00-24.00   sec  5.96 MBytes  50.0 Mbits/sec  0.225 ms     0/4315 (0%)
[ 5] 24.00-25.00   sec  5.96 MBytes  50.0 Mbits/sec  0.406 ms     0/4315 (0%)
[ 5] 25.00-26.00   sec  5.96 MBytes  50.0 Mbits/sec  0.230 ms     0/4319 (0%)
[ 5] 26.00-27.00   sec  5.95 MBytes  50.0 Mbits/sec  0.190 ms     0/4312 (0%)
[ 5] 27.00-28.00   sec  5.97 MBytes  50.0 Mbits/sec  0.326 ms     0/4320 (0%)
[ 5] 28.00-29.00   sec  5.96 MBytes  50.0 Mbits/sec  0.311 ms     0/4316 (0%)
[ 5] 29.00-30.00   sec  5.96 MBytes  50.0 Mbits/sec  0.156 ms     0/4314 (0%)
[ 5] 30.00-30.05   sec   266 KBytes  43.2 Mbits/sec  0.404 ms     0/188 (0%)
-----
[ ID] Interval      Transfer    Bitrate      Jitter      Lost/Total Datagrams
[ 5]  0.00-30.05   sec  168 MBytes  47.0 Mbits/sec  0.404 ms    7502/129485 (5.8%) receiver

```

Рисунок 23 – Джиттер RED

Анализ результатов выявил ключевое отличие RED от FIFO — наличие потерянных пакетов. Алгоритм отбросил 7 502 пакета из 129 485 переданных, что составляет 5,79 % от общего объёма трафика. Данные потери не являются следствием переполнения очереди, а представляют собой результат упреждающего вероятностного отбрасывания — основного механизма работы RED. При достижении очередью минимального порога алгоритм начинает случайным образом отбрасывать входящие пакеты, сигнализируя протоколам верхнего уровня (TCP) о необходимости снижения скорости передачи. Поскольку тестирование проводилось с использованием UDP-трафика, который не реагирует на потери, снижения интенсивности поступления пакетов не происходило, и алгоритм продолжал отбрасывание на протяжении всего эксперимента.

Снижение пропускной способности до 47,0 Мбит/с (потеря 3 Мбит/с относительно входной скорости) является прямым следствием отбрасывания пакетов. Джиттер на уровне 0,404 мс оказался незначительно ниже, чем у FIFO (0,427 мс), что объясняется уменьшением средней длины очереди благодаря превентивным сбросам — пакеты реже накапливаются в буфере и быстрее покидают его.

Таким образом, алгоритм RED выполняет свою основную функцию — предотвращает полное заполнение очереди ценой управляемых потерь. В условиях UDP-нагрузки это приводит к снижению пропускной способности без существенного выигрыша в джиттере, однако в сетях с TCP-трафиком вероятностное отбрасывание позволяет заблаговременно сигнализировать источникам о перегрузке, предотвращая лавинообразный рост задержек.

1.9.3 Тестирование CoDel

Тестирование алгоритма CoDel (Controlled Delay) проводилось в тех же условиях: длительность 30 секунд, UDP-трафик с номинальной скоростью 50 Мбит/с. Параметры алгоритма: целевая задержка — 5 мс, интервал мониторинга — 100 мс. Результаты измерений представлены на рисунке 24.

Анализ результатов демонстрирует эффективность алгоритма CoDel в условиях заданной нагрузки. Пропускная способность составила 49,9 Мбит/с, что идентично показателю FIFO и соответствует номинальной скорости генерации трафика. Потери пакетов отсутствуют, что свидетельствует о том, что время пребывания пакетов в очереди (sojourn time) ни разу не превысило целевое значение 5 мс в течение интервала 100 мс — условие входа в режим отбрасывания не было выполнено.

```

Server listening on 5201
-----
Accepted connection from 10.0.1.10, port 50710
[ 5] local 10.0.2.10 port 5201 connected to 10.0.1.10 port 38698
[ ID] Interval      Transfer    Bitrate      Jitter      Lost/Total Datagrams
[ 5]  0.00-1.00    sec  5.67 MBytes  47.5 Mbits/sec  0.358 ms  0/4105 (0%)
[ 5]  1.00-2.00    sec  5.97 MBytes  50.0 Mbits/sec  0.163 ms  0/4322 (0%)
[ 5]  2.00-3.00    sec  5.95 MBytes  50.0 Mbits/sec  0.251 ms  0/4311 (0%)
[ 5]  3.00-4.00    sec  5.96 MBytes  50.0 Mbits/sec  0.299 ms  0/4316 (0%)
[ 5]  4.00-5.00    sec  5.97 MBytes  50.0 Mbits/sec  0.307 ms  0/4320 (0%)
[ 5]  5.00-6.00    sec  5.96 MBytes  50.0 Mbits/sec  0.436 ms  0/4316 (0%)
[ 5]  6.00-7.00    sec  5.96 MBytes  50.0 Mbits/sec  0.204 ms  0/4318 (0%)
[ 5]  7.00-8.00    sec  5.96 MBytes  50.0 Mbits/sec  0.262 ms  0/4316 (0%)
[ 5]  8.00-9.00    sec  5.96 MBytes  50.0 Mbits/sec  0.414 ms  0/4316 (0%)
[ 5]  9.00-10.00   sec  5.96 MBytes  50.0 Mbits/sec  0.368 ms  0/4317 (0%)
[ 5] 10.00-11.00   sec  5.96 MBytes  50.0 Mbits/sec  0.401 ms  0/4314 (0%)
[ 5] 11.00-12.00   sec  5.96 MBytes  50.0 Mbits/sec  0.173 ms  0/4318 (0%)
[ 5] 12.00-13.00   sec  5.96 MBytes  50.0 Mbits/sec  0.272 ms  0/4315 (0%)
[ 5] 13.00-14.00   sec  5.96 MBytes  50.0 Mbits/sec  0.385 ms  0/4313 (0%)
[ 5] 14.00-15.00   sec  5.96 MBytes  50.0 Mbits/sec  0.187 ms  0/4316 (0%)
[ 5] 15.00-16.00   sec  5.96 MBytes  49.8 Mbits/sec  0.277 ms  0/4316 (0%)
[ 5] 16.00-17.00   sec  5.97 MBytes  50.2 Mbits/sec  0.151 ms  0/4321 (0%)
[ 5] 17.00-18.00   sec  5.96 MBytes  50.0 Mbits/sec  0.282 ms  0/4316 (0%)
[ 5] 18.00-19.00   sec  5.96 MBytes  50.0 Mbits/sec  0.345 ms  0/4313 (0%)
[ 5] 19.00-20.00   sec  5.96 MBytes  50.0 Mbits/sec  0.320 ms  0/4319 (0%)
[ 5] 20.00-21.00   sec  5.96 MBytes  50.0 Mbits/sec  0.316 ms  0/4313 (0%)
[ 5] 21.00-22.00   sec  5.96 MBytes  50.0 Mbits/sec  0.421 ms  0/4319 (0%)
[ 5] 22.00-23.00   sec  5.95 MBytes  50.0 Mbits/sec  0.334 ms  0/4312 (0%)
[ 5] 23.00-24.00   sec  5.96 MBytes  50.0 Mbits/sec  0.408 ms  0/4319 (0%)
[ 5] 24.00-25.00   sec  5.96 MBytes  50.0 Mbits/sec  0.341 ms  0/4318 (0%)
[ 5] 25.00-26.00   sec  5.96 MBytes  50.0 Mbits/sec  0.342 ms  0/4313 (0%)
[ 5] 26.00-27.00   sec  5.96 MBytes  50.0 Mbits/sec  0.267 ms  0/4318 (0%)
[ 5] 27.00-28.00   sec  5.96 MBytes  50.0 Mbits/sec  0.458 ms  0/4313 (0%)
[ 5] 28.00-29.00   sec  5.97 MBytes  50.0 Mbits/sec  0.174 ms  0/4320 (0%)
[ 5] 29.00-30.00   sec  5.95 MBytes  49.9 Mbits/sec  0.473 ms  0/4309 (0%)
[ 5] 30.00-30.05   sec   297 KBytes  45.1 Mbits/sec  0.355 ms  0/210 (0%)
-----
[ ID] Interval      Transfer    Bitrate      Jitter      Lost/Total Datagrams
[ 5]  0.00-30.05   sec  179 MBytes  49.9 Mbits/sec  0.355 ms  0/129482 (0%) receiver
-----

```

Рисунок 24 – Джиттер CoDel

Ключевым результатом является снижение джиттера до 0,355 мс, что на 16,9 % ниже показателя FIFO (0,427 мс). Данное улучшение объясняется принципом работы CoDel: алгоритм непрерывно контролирует задержку каждого

пакета и при возникновении тенденции к росту времени ожидания превентивно ограничивает наполнение очереди. Даже не переходя в активный режим отбрасывания, сам факт мониторинга задержки и наличие механизма реагирования способствует поддержанию очереди в состоянии, близком к оптимальному.

Таким образом, алгоритм CoDel при заданной нагрузке обеспечивает такую же пропускную способность и нулевые потери, как FIFO, но превосходит его по показателю джиттера. Это подтверждает эффективность подхода, основанного на контроле задержки, а не длины очереди: алгоритм не вмешивается в обработку трафика без необходимости, но готов оперативно реагировать при ухудшении характеристик. Отсутствие настраиваемых порогов (в отличие от RED) делает CoDel универсальным решением, не требующим адаптации параметров под конкретные условия сети.

1.9.4 Тестирование ARIMA

Тестирование алгоритма ARIMA-pacing проводилось в тех же условиях: длительность 30 секунд, UDP-трафик с номинальной скоростью 50 Мбит/с. Параметры модели: ARIMA(2,1,1), окно обучения — 100 наблюдений, пересчёт коэффициентов — каждые 10 пакетов, начальный интервал pacing — 1000 мкс. Результаты измерений представлены на рисунке 25.

Анализ результатов показывает, что ARIMA-pacing обеспечивает наилучший показатель джиттера среди всех протестированных алгоритмов — 0,318 мс, что на 25,5 % ниже FIFO (0,427 мс), на 21,3 % ниже RED (0,404 мс) и на 10,4 % ниже CoDel (0,355 мс). При этом пропускная способность сохранена на уровне 49,9 Мбит/с, потери пакетов отсутствуют.

Достигнутое снижение джиттера является прямым следствием механизма pacing — искусственного выравнивания интервалов между отправляемыми пакетами. В то время как FIFO, RED и CoDel выпускают пакеты по мере возможности (с ограничениями в случае AQM), ARIMA-pacing на основе прогноза модели формирует равномерный выходной поток. Модель

ARIMA(2,1,1), обучаясь на временном ряде интервалов между входящими пакетами, предсказывает момент прибытия следующего пакета, что позволяет заблаговременно скорректировать rasing-интервал. Адаптивный механизм плавно подстраивает темп выпуска: при учащении входящих пакетов интервал сокращается (но не ниже 50 мкс), при замедлении — увеличивается (до 2000 мкс).

```

Server listening on 5201
-----
Accepted connection from 10.0.1.10, port 36308
[ 5] local 10.0.2.10 port 5201 connected to 10.0.1.10 port 38311
[ ID] Interval      Transfer    Bitrate      Jitter    Lost/Total Datagrams
[ 5]  0.00-1.00    sec  5.65 MBytes 47.4 Mbits/sec 0.394 ms  0/4092 (0%)
[ 5]  1.00-2.00    sec  5.96 MBytes 50.0 Mbits/sec 0.379 ms  0/4319 (0%)
[ 5]  2.00-3.00    sec  5.95 MBytes 49.9 Mbits/sec 0.316 ms  0/4311 (0%)
[ 5]  3.00-4.00    sec  5.97 MBytes 50.0 Mbits/sec 0.307 ms  0/4321 (0%)
[ 5]  4.00-5.00    sec  5.96 MBytes 50.0 Mbits/sec 0.324 ms  0/4317 (0%)
[ 5]  5.00-6.00    sec  5.96 MBytes 50.0 Mbits/sec 0.210 ms  0/4318 (0%)
[ 5]  6.00-7.00    sec  5.96 MBytes 50.0 Mbits/sec 0.369 ms  0/4316 (0%)
[ 5]  7.00-8.00    sec  5.96 MBytes 50.0 Mbits/sec 0.366 ms  0/4314 (0%)
[ 5]  8.00-9.00    sec  5.97 MBytes 50.1 Mbits/sec 0.286 ms  0/4320 (0%)
[ 5]  9.00-10.00   sec  5.96 MBytes 50.0 Mbits/sec 0.411 ms  0/4314 (0%)
[ 5] 10.00-11.00   sec  5.96 MBytes 50.0 Mbits/sec 0.415 ms  0/4314 (0%)
[ 5] 11.00-12.00   sec  5.96 MBytes 50.0 Mbits/sec 0.334 ms  0/4316 (0%)
[ 5] 12.00-13.00   sec  5.96 MBytes 50.0 Mbits/sec 0.356 ms  0/4317 (0%)
[ 5] 13.00-14.00   sec  5.96 MBytes 50.0 Mbits/sec 0.319 ms  0/4315 (0%)
[ 5] 14.00-15.00   sec  5.96 MBytes 50.0 Mbits/sec 0.358 ms  0/4319 (0%)
[ 5] 15.00-16.00   sec  5.96 MBytes 50.0 Mbits/sec 0.273 ms  0/4317 (0%)
[ 5] 16.00-17.00   sec  5.96 MBytes 50.0 Mbits/sec 0.297 ms  0/4317 (0%)
[ 5] 17.00-18.00   sec  5.96 MBytes 50.0 Mbits/sec 0.334 ms  0/4314 (0%)
[ 5] 18.00-19.00   sec  5.97 MBytes 50.0 Mbits/sec 0.406 ms  0/4320 (0%)
[ 5] 19.00-20.00   sec  5.96 MBytes 50.0 Mbits/sec 0.287 ms  0/4314 (0%)
[ 5] 20.00-21.00   sec  5.96 MBytes 49.9 Mbits/sec 0.324 ms  0/4313 (0%)
[ 5] 21.00-22.00   sec  5.96 MBytes 50.1 Mbits/sec 0.289 ms  0/4317 (0%)
[ 5] 22.00-23.00   sec  5.96 MBytes 50.0 Mbits/sec 0.278 ms  0/4316 (0%)
[ 5] 23.00-24.00   sec  5.96 MBytes 50.0 Mbits/sec 0.324 ms  0/4317 (0%)
[ 5] 24.00-25.00   sec  5.96 MBytes 50.0 Mbits/sec 0.286 ms  0/4316 (0%)
[ 5] 25.00-26.00   sec  5.96 MBytes 50.0 Mbits/sec 0.275 ms  0/4318 (0%)
[ 5] 26.00-27.00   sec  5.96 MBytes 50.0 Mbits/sec 0.385 ms  0/4316 (0%)
[ 5] 27.00-28.00   sec  5.96 MBytes 50.0 Mbits/sec 0.296 ms  0/4315 (0%)
[ 5] 28.00-29.00   sec  5.96 MBytes 50.0 Mbits/sec 0.200 ms  0/4315 (0%)
[ 5] 29.00-30.00   sec  5.96 MBytes 50.0 Mbits/sec 0.446 ms  0/4318 (0%)
[ 5] 30.00-30.06   sec   305 KBytes 43.7 Mbits/sec 0.318 ms  0/216 (0%)
-----
[ ID] Interval      Transfer    Bitrate      Jitter    Lost/Total Datagrams
[ 5]  0.00-30.06   sec  179 MBytes 49.9 Mbits/sec 0.318 ms  0/129482 (0%) receiver
-----

```

Рисунок 25 – Джиттер ARIMA

Отсутствие потерь объясняется двумя факторами. Во-первых, объёма буферной памяти (128 МБ) достаточно для компенсации неравномерности поступления пакетов без переполнения. Во-вторых, алгоритм не применяет принудительное отбрасывание (в отличие от RED и CoDel), а реализует механизм сглаживания — временное накопление пакетов в очереди с последующим равномерным выпуском. При росте очереди свыше трёх пакетов rasing временно

отключается, что предотвращает создание избыточной задержки и одновременно защищает от переполнения.

Вычислительные накладные расходы, связанные с оценкой коэффициентов ARIMA через LU-разложение матрицы Теплица, не оказали заметного влияния на производительность благодаря оптимизации: память под матрицы выделена однократно при инициализации, пересчёт модели выполняется не чаще чем раз в 10 пакетов. С учётом того, что при скорости 50 Мбит/с средний межпакетный интервал составляет порядка 200–240 мкс, а расчёт модели занимает единицы микросекунд, вычислительная нагрузка остаётся пренебрежимо малой относительно времени обработки пакета.

Таким образом, алгоритм ARIMA-pacing демонстрирует наилучшие показатели джиттера среди всех рассмотренных алгоритмов при сохранении полной пропускной способности и отсутствии потерь. Способность адаптироваться к характеру трафика без статической настройки пороговых значений делает данный подход универсальным решением для сетей с динамической нагрузкой, а использование методов прогнозирования временных рядов открывает перспективы для дальнейшего совершенствования алгоритмов активного управления очередью.

2 Охрана труда

2.1 Описание негативных факторов, воздействующих на разработчика и способы их устранения

Взаимодействие с компьютером происходит в условиях ограниченной двигательной активности и длительного статического напряжения, что приводит к негативным последствиям, таким как:

- снижение общей работоспособности;
- ухудшение зрения;
- развитие гиподинамии;
- повышение нервно-психического напряжения;
- хроническое утомление и переутомление.

Для уменьшения влияния перечисленных факторов на человека необходимо соблюдать следующие правила работы за компьютером:

- соблюдать режим труда и отдыха с регламентированными перерывами;
- рационально организовывать рабочее пространство;
- соблюдать правила эксплуатации электрооборудования;
- выполнять требования пожарной безопасности.

2.1.1 Организация рабочего места

Для обеспечения комфортной и безопасной работы за персональным компьютером предъявляются следующие требования к организации рабочего места:

- расстояние от глаз до экрана монитора должно составлять не менее 500 мм;
- монитор следует располагать ниже уровня глаз;
- угол между линией взгляда прямо и направлением на центр экрана должен составлять приблизительно 30 градусов;
- высота рабочего стола — от 620 до 820 мм;

– пространство для ног должно иметь высоту не менее 600 мм, глубину на уровне колен — не менее 450 мм, глубину на уровне вытянутых ног — не менее 650 мм.

На рисунке 26 изображена подходящая конфигурация рабочего места, которая отвечает указанным выше требованиям.

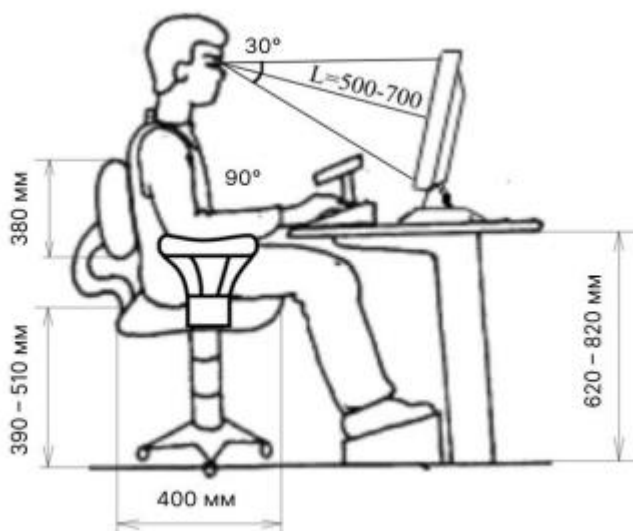


Рисунок 26 – Требуемая организация рабочего места

Рабочее место с дисплеем должно обеспечивать оператору возможность выполнения работ в положении сидя без перегрузки костно-мышечной системы.

При правильной организации рабочего места необходимо обеспечить корректную посадку пользователя:

- запястья находятся в прямом положении;
- плечи расслаблены;
- локти располагаются перпендикулярно полу (угол в локтевых суставах — 90–100 градусов).

2.1.2 Пожарная безопасность

При эксплуатации вычислительной техники наиболее частыми причинами пожаров являются короткое замыкание, воспламенение горючих материалов

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.31.001 ПЗ

Лист

57

вблизи электроустройств и большие переходные сопротивления в контактных соединениях.

Меры профилактики проводятся согласно Федеральному закону № 69-ФЗ «О пожарной безопасности». Для обеспечения пожарной безопасности необходимо:

- содержать проходы и запасные выходы свободными для эвакуации;
- обеспечивать свободный доступ к средствам пожаротушения;
- оборудовать систему вентиляции огнезадерживающими клапанами с автоматическим отключением при пожаре;
- установить и периодически проверять автоматическую пожарную сигнализацию и средства пожаротушения.

2.1.3 Электробезопасность

Требования предъявляются согласно ГОСТ 12.1.019-2017 «Электробезопасность». Персональный компьютер питается от сети переменного тока частотой 50 Гц напряжением 230 В, что опасно для жизни. При работе необходимо соблюдать следующие правила:

- все устройства должны питаться от одной фазы электрической сети;
- подключение должно производиться от розеток с заземляющими контактами;
- розетки должны иметь маркировку по напряжению;
- значение номинального напряжения наносится яркой краской крупными символами (высотой не менее 50 мм) на стене или щите возле розетки.

Необходимо регулярно проверять состояние проводки, выключателей и розеток. При обнаружении неисправности оборудование немедленно отключается до полного устранения дефектов.

2.1.4 Режим труда и отдыха

Пользователь обязан соблюдать режим труда и отдыха в зависимости от продолжительности, вида и категории трудовой деятельности. Виды работ с ПК делятся на группы:

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		58

– группа А — считывание информации с дисплея по предварительному запросу;

– группа Б — ввод информации;

– группа В — творческая работа в режиме диалога с ПК.

При выполнении работ разных видов за основную принимается та деятельность, которая занимает не менее 50% рабочего времени в течение смены. Категории тяжести и напряжённости работы с ПК и соответствующие им времена перерывов приведены в таблице 3.

Таблица 3— Время перерывов

Категория работы с ПЭВМ или ВДТ	Уровень нагрузки за рабочую смену при видах работы с ВДТ			Суммарное время регламентированных перерывов, мин	
	группа А, количество знаков	группа Б, количество знаков	группа В, количество часов	при 8-часовой смене	При 12-часовой смене
1	До 20000	До 15000	До 2	50	80
2	До 40000	До 30000	До 4	70	110
3	До 60000	До 40000	До 6	90	140

3 Стартап проект

3.1 Описание существующей проблемы на рынке

Проблема, которую решает проект, заключается в отсутствии эффективных и масштабируемых решений для анализа темпорально-корреляционных свойств сетевого трафика с целью повышения характеристик качества обслуживания QoS/QoE в условиях высокой нестабильности сетевой среды и плохой сходимости выборочных характеристик трафика. Несмотря на стремительное развитие сетевых технологий, распределённых вычислительных систем и сервисов реального времени, современные сетевые устройства L2/L3-уровней в большинстве случаев используют классические алгоритмы обработки трафика, ориентированные преимущественно на усреднённые параметры нагрузки и не учитывающие сложную временную структуру сетевых потоков.

В современных условиях цифровизации особую значимость приобретают сервисы, предъявляющие повышенные требования к стабильности сетевого соединения и качеству передачи данных. К таким сервисам относятся системы видеоконференцсвязи, IP-телефония, потоковое мультимедиа, облачные игровые платформы, распределённые интерактивные сервисы, системы промышленного интернета вещей (IIoT), MQTT-ориентированные сети и критически важные системы телеметрии. Для корректного функционирования подобных приложений недостаточно обеспечить только высокую пропускную способность канала связи. Ключевое значение приобретают стабильность задержек передачи пакетов, минимизация jitter и снижение вероятности потерь данных.

Одной из наиболее серьёзных проблем современных сетей является джиттер — вариация времени доставки пакетов. Даже при относительно низкой средней задержке значительные колебания времени передачи данных способны приводить к ухудшению качества мультимедийных сервисов, потере синхронизации потоков, буферизации, искажению аудио- и видеоданных и снижению общего качества пользовательского опыта. Особенно остро данная

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		60

проблема проявляется в условиях нестабильных и ненадёжных сетей, где характеристики трафика обладают сложной стохастической природой и выраженными корреляционными зависимостями.

Сложность обеспечения стабильного QoS/QoE обусловлена высокой динамичностью сетевого трафика. В реальных условиях нагрузки изменяются во времени, формируя сложные нелинейные зависимости между интенсивностью потока, задержками, потерями пакетов, загруженностью очередей и пропускной способностью каналов связи. При этом современные алгоритмы диспетчеризации очередей и traffic shaping в большинстве существующих маршрутизаторов и коммутаторов ориентированы на реактивные методы управления трафиком и не учитывают темпоральные и корреляционные свойства сетевых потоков.

Дополнительной проблемой является плохая сходимость выборочных характеристик качества обслуживания в условиях самоподобного и коррелированного трафика. В таких сетях классические статистические методы анализа оказываются недостаточно точными, поскольку усреднённые показатели не отражают реальное поведение системы во времени. Это приводит к снижению эффективности алгоритмов QoS и невозможности своевременного прогнозирования перегрузок сети.

Особую актуальность проблема приобретает в рамках IoT и MQTT-ориентированных сетей, функционирующих в условиях крайне ненадёжной коммуникационной среды. В подобных системах требуется обеспечение устойчивой передачи телеметрической информации при ограниченных вычислительных ресурсах, высокой вероятности потерь пакетов и нестабильных каналах связи. Существующие реализации сетевого оборудования зачастую не обеспечивают необходимого уровня адаптивности и интеллектуального управления потоками данных в подобных условиях эксплуатации.

Целью данного проекта является разработка блока анализа темпорально-корреляционных свойств сетевого трафика, обладающего плохой сходимостью выборочных характеристик качества обслуживания, обеспечивающего

улучшение показателей QoS/QoE по сравнению с существующими реализациями сетевого оборудования L2/L3-уровней. Разрабатываемое решение представляет собой программно-аппаратный комплекс, предназначенный для интеллектуального анализа временных и корреляционных характеристик сетевого трафика с последующей адаптацией механизмов управления очередями и обработки пакетов.

Ключевой задачей проекта является создание масштабируемой архитектуры, способной функционировать как в виде программного комплекса для существующего сетевого оборудования, так и в виде аппаратного ускорителя для открытых аппаратных платформ. В рамках проекта предполагается использование современных методов packet scheduling, active queue management, корреляционного анализа и статистической обработки временных рядов сетевого трафика. Реализация проекта позволит перейти от классических реактивных механизмов управления трафиком к интеллектуальным проактивным системам, способным прогнозировать возникновение перегрузок сети, адаптироваться к изменяющимся условиям передачи данных и обеспечивать более стабильное качество сетевого обслуживания в условиях высокой динамичности и ненадёжности сетевой среды.

3.2 Описание решения проблемы и продуктового решения

Разрабатываемый программно-аппаратный комплекс представляет собой блок анализа темпорально-корреляционных свойств сетевого трафика, предназначенный для повышения эффективности механизмов QoS/QoE в сетях передачи данных с высокой вариативностью временных характеристик. Основной задачей системы является анализ временной структуры сетевого трафика, выявление корреляционных зависимостей между параметрами передачи данных и интеллектуальное управление обработкой пакетов с целью минимизации jitter, задержек и потерь информации.

В основе предлагаемого решения лежит многопоточная архитектура программного маршрутизатора, реализованного на языке C++20 для

операционной системы Linux Debian с использованием raw sockets. Система обеспечивает низкоуровневую обработку сетевых пакетов в userspace и поддерживает двунаправленную маршрутизацию трафика с разделением потоков приёма и передачи данных. Такой подход позволяет минимизировать взаимные блокировки потоков и обеспечить стабильную обработку сетевого трафика в режиме реального времени.

Архитектура комплекса предусматривает использование четырёх независимых потоков:

- RX поток направления $A \rightarrow B$;
- TX поток направления $A \rightarrow B$;
- RX поток направления $B \rightarrow A$;
- TX поток направления $B \rightarrow A$.

Разделение потоков RX и TX позволяет снизить конкуренцию между потоками, повысить стабильность обработки пакетов и обеспечить более эффективную работу алгоритмов управления очередями.

Центральным элементом системы является модуль диспетчеризации очередей IScheduler, реализующий интеллектуальное управление пакетами. Данный модуль обеспечивает:

- добавление пакетов в очередь;
- извлечение пакетов для передачи;
- анализ времени нахождения пакетов в очереди;
- контроль использования памяти;
- управление политиками сброса пакетов;
- снижение jitter и latency;
- реализацию алгоритмов active queue management.

В рамках проекта предусмотрена поддержка современных алгоритмов обработки трафика:

- FIFO;
- RED;

- CoDel;
- пользовательских алгоритмов анализа корреляционных свойств трафика.

Особенностью решения является возможность динамического подключения алгоритмов обработки очередей через .so библиотеки. Для загрузки модулей используются механизмы dlopen, dlsym и dlclose, что обеспечивает гибкость архитектуры и позволяет адаптировать систему под различные прикладные задачи без изменения ядра программного маршрутизатора.

Для анализа временных характеристик сетевого трафика используется сущность Packet, содержащая:

- raw packet data;
- размер пакета;
- временную метку помещения пакета в очередь.

Использование временных меток позволяет реализовать анализ задержек передачи, оценку jitter, исследование временных рядов и построение корреляционных зависимостей между параметрами сетевого трафика.

Дополнительно в системе реализован модуль MemoryTracker, предназначенный для контроля использования heap memory очередями маршрутизатора. При достижении установленного лимита памяти алгоритмы Scheduler принимают решения о сбросе пакетов в соответствии с выбранной политикой управления перегрузкой сети.

Разрабатываемый комплекс ориентирован на применение в:

- системах QoS/QoE management;
- IoT-сетях;
- MQTT-ориентированных системах;
- высоконагруженных распределённых сервисах;
- мультимедийных потоковых системах;
- сетевых исследовательских комплексах;
- программно-определяемых сетях.

В рамках проекта предусматривается исследование и применение инструментальных средств моделирования сетевой активности, включая GNS3, а также библиотек статистической и частотно-временной обработки данных, например GNU Scientific Library (GSL). Дополнительно предполагается использование библиотек packet inspection и анализа сетевых протоколов.

Особое внимание уделяется реализации механизмов интеллектуального анализа темпорально-корреляционных свойств трафика, позволяющих выявлять закономерности изменения параметров сети во времени и прогнозировать возникновение перегрузок. Такой подход создаёт предпосылки для перехода от реактивных методов управления сетью к проактивным алгоритмам адаптивного управления трафиком.

Основные характеристики разрабатываемого решения:

- язык разработки — C++20;
- операционная система — Linux Debian;
- использование raw sockets;
- многопоточная архитектура;
- поддержка packet scheduling;
- реализация active queue management;
- анализ temporal/correlation characteristics;
- контроль jitter и latency;
- динамическая загрузка алгоритмов;
- масштабируемая модульная архитектура;
- поддержка работы в условиях нестабильных сетей;
- совместимость с IoT и MQTT-инфраструктурой.

Результатом проекта станет программно-аппаратный блок анализа темпорально-корреляционных свойств трафика, обеспечивающий более эффективное управление сетевыми потоками и повышение качества обслуживания QoS/QoE по сравнению с традиционными решениями сетевого оборудования L2/L3-уровней.

3.3Объём, структура и развитие рынка

В целевые сегменты рынка входят телекоммуникационное направление, сфера промышленного интернета вещей и рынок облачных мультимедийных сервисов.

Телекоммуникационное и сетевое направление. Согласно данным аналитических агентств IDC, MarketsandMarkets и Grand View Research, мировой рынок систем анализа сетевого трафика, мониторинга производительности сети и решений для управления QoS/QoE демонстрирует устойчивый рост. По прогнозам аналитиков, к 2028 году объём мирового рынка network traffic analytics и network performance monitoring может превысить 8–10 млрд долларов при среднегодовом темпе роста более 15%. Основными драйверами роста являются увеличение объёмов передаваемых данных, развитие облачных вычислений, распространение сервисов реального времени и рост требований к стабильности передачи информации.

В современных условиях всё большее значение приобретают приложения, чувствительные к задержкам и джиттеру. К таким системам относятся:

- видеоконференцсвязь;
- IP-телефония;
- потоковое мультимедиа;
- облачные игровые платформы;
- VR/AR-сервисы;
- распределённые интерактивные приложения.

Для подобных решений критически важны не только высокая пропускная способность сети, но и стабильность временных характеристик передачи данных. Даже кратковременные колебания задержек могут приводить к ухудшению качества пользовательского опыта, потере синхронизации потоков, буферизации и искажению аудио- и видеоданных. В связи с этим возрастает спрос на интеллектуальные системы управления сетевым трафиком, способные

анализировать временные характеристики сетевых потоков и адаптивно управлять очередями передачи данных.

Разрабатываемый блок анализа темпорально-корреляционных свойств трафика ориентирован на данный сегмент рынка и предназначен для повышения эффективности обработки сетевых потоков за счёт интеллектуального анализа временных характеристик трафика и адаптивного управления очередями.

Направление ПоТ и MQTT-инфраструктуры. Одним из наиболее перспективных сегментов применения технологии является промышленный интернет вещей (ПоТ). Согласно данным Statista и Fortune Business Insights, мировой рынок ПоТ к 2030 году может превысить 1 трлн долларов при среднегодовом темпе роста более 20%.

Современные ПоТ-системы функционируют в условиях:

- нестабильных каналов связи;
- ограниченной пропускной способности;
- высокой плотности телеметрических сообщений;
- необходимости передачи данных в реальном времени.

Особую роль в ПоТ-инфраструктурах играют MQTT-ориентированные системы обмена сообщениями, широко применяемые в:

- промышленной автоматизации;
- системах мониторинга;
- телеметрии;
- энергетике;
- транспортных системах;
- концепции «умного города».

Для подобных сетей критически важными являются:

- минимизация jitter;
- стабильность задержек;
- устойчивость передачи данных;
- прогнозирование перегрузок сети.

Существующие сетевые решения зачастую не обеспечивают необходимого уровня адаптивности к временным изменениям сетевого трафика, особенно в условиях ненадёжной коммуникационной среды. Разрабатываемый программно-аппаратный комплекс ориентирован на решение данной проблемы за счёт применения методов темпорально-корреляционного анализа и интеллектуального управления пакетными очередями.

Направление облачных и мультимедийных сервисов. Дополнительным перспективным направлением является рынок облачных мультимедийных платформ и сервисов реального времени. По данным Cisco и Gartner, более 80% мирового интернет-трафика в ближайшие годы будет приходиться на потоковые мультимедийные данные. Одновременно растёт количество пользователей:

- видеосервисов;
- облачного гейминга;
- VR/AR-платформ;
- интерактивных распределённых приложений.

Для таких систем стабильность передачи данных и минимизация вариации задержек имеют критически важное значение. Даже кратковременные скачки jitter способны вызывать:

- потерю синхронизации;
- буферизацию;
- ухудшение качества изображения и звука;
- снижение качества пользовательского опыта (QoE).

Разрабатываемая система анализа темпорально-корреляционных свойств трафика позволяет повысить стабильность передачи данных за счёт интеллектуального управления очередями и прогнозирования перегрузок сети.

3.4 Анализ конкурентов

На современном рынке систем управления сетевым трафиком и обеспечения QoS/QoE существует большое количество решений, ориентированных на packet scheduling, traffic shaping и мониторинг сетевой

активности. Однако большинство существующих платформ не обеспечивают полноценного анализа темпорально-корреляционных характеристик сетевого трафика и ориентированы преимущественно на классические методы обработки потоков данных.

Одним из наиболее распространённых решений является Linux Traffic Control (tc), входящий в состав ядра Linux. Данный инструмент поддерживает различные алгоритмы диспетчеризации очередей, включая:

- FIFO;
- RED;
- SFQ;
- CoDel;
- FQ-CoDel.

Основными преимуществами tc являются высокая производительность и интеграция с Linux-системами. Однако настройка подобных механизмов требует высокой квалификации администратора, а возможности анализа временных и корреляционных свойств трафика остаются ограниченными. Кроме того, архитектура tc недостаточно гибка для подключения экспериментальных алгоритмов анализа сетевых потоков.

Широкое распространение получили программные маршрутизаторы pfSense и OPNsense, обеспечивающие функциональность:

- межсетевого экранирования;
- traffic shaping;
- мониторинга сетевой активности;
- управления QoS.

Данные решения активно применяются в корпоративных сетях и обладают развитым пользовательским интерфейсом. Однако механизмы QoS в них преимущественно основаны на статических правилах обработки трафика и не используют интеллектуальный анализ временной структуры сетевых потоков.

В сегменте корпоративных решений лидирующие позиции занимают Cisco и Juniper Networks, предлагающие аппаратно-программные платформы с поддержкой:

- packet scheduling;
- traffic shaping;
- deep packet inspection;
- SLA management;
- интеллектуальной маршрутизации.

Несмотря на высокую производительность и развитые механизмы QoS, данные системы обладают рядом недостатков:

- высокая стоимость внедрения;
- закрытая архитектура;
- ограниченные возможности модификации алгоритмов;
- сложность интеграции исследовательских механизмов анализа трафика.

Дополнительно значительная часть подобных решений ориентирована на крупные дата-центры и операторские сети, что ограничивает их применение в научных исследованиях и специализированных ПоТ-инфраструктурах.

В области программно-определяемых сетей широкое распространение получили Open vSwitch и DPDK. Эти технологии обеспечивают высокоскоростную обработку пакетов и активно применяются в cloud- и edge-инфраструктурах. Однако их основная задача связана с ускорением packet processing, тогда как задачи интеллектуального анализа темпорально-корреляционных свойств трафика и прогнозирования QoS/QoE реализованы ограниченно.

Среди исследовательских решений можно выделить проекты, использующие:

- машинное обучение для traffic prediction;
- анализ временных рядов;
- статистическое моделирование сетевого поведения;

- корреляционный анализ параметров сети.

Однако большинство подобных систем существуют в виде экспериментальных платформ и не обладают универсальной архитектурой, пригодной для интеграции в реальные сетевые инфраструктуры.

Разрабатываемый программно-аппаратный блок анализа темпорально-корреляционных свойств трафика отличается от существующих решений рядом ключевых особенностей:

- поддержка анализа temporal/correlation characteristics сетевого трафика;
- интеллектуальное управление очередями;
- ориентация на снижение jitter и стабилизацию QoS/QoE;
- модульная архитектура;
- поддержка динамической загрузки алгоритмов через .so библиотеки;
- возможность интеграции в существующее сетевое оборудование;
- поддержка работы в нестабильных сетях и IoT-инфраструктурах;
- использование raw sockets и многопоточной обработки пакетов.

Ключевым преимуществом проекта является сочетание исследовательской гибкости и практической применимости. В отличие от большинства существующих решений, система позволяет:

- экспериментально подключать новые алгоритмы packet scheduling;
- исследовать корреляционные зависимости сетевого трафика;
- анализировать временные ряды задержек;
- адаптивно управлять очередями в режиме реального времени.

Анализ конкурентной среды показывает, что на рынке практически отсутствуют решения, одновременно сочетающие:

- анализ темпорально-корреляционных характеристик;
- адаптивное управление QoS/QoE;
- модульную архитектуру;
- поддержку экспериментальных алгоритмов;
- возможность работы в нестабильных IoT-сетях;

– масштабируемость и гибкость интеграции.

Это создаёт перспективную рыночную нишу для разрабатываемого программно-аппаратного комплекса, который способен занять промежуточное положение между высокоспециализированными корпоративными решениями и исследовательскими платформами сетевого анализа.

3.5 Бизнес-модель проекта

Модель монетизации проекта основана на продаже программно-аппаратного комплекса для анализа темпорально-корреляционных свойств сетевого трафика, лицензировании программного обеспечения, а также предоставлении услуг технической поддержки и сопровождения. Продукт может поставляться как в виде самостоятельного программного комплекса для интеграции в существующее сетевое оборудование, так и в виде специализированного аппаратного ускорителя для высоконагруженных сетевых инфраструктур.

Основными источниками дохода проекта являются:

- продажа лицензий на программное обеспечение;
- поставка программно-аппаратных комплексов;
- внедрение и интеграция решения в существующую инфраструктуру заказчика;
- техническая поддержка и сопровождение;
- разработка специализированных модулей под задачи заказчика;
- консалтинговые услуги в области оптимизации QoS/QoE.

Ключевыми каналами продаж являются:

- прямые B2B-продажи;
- партнёрские интеграции;
- поставки через системных интеграторов;
- лицензирование для разработчиков сетевого оборудования;
- внедрение в корпоративные и промышленные инфраструктуры.

Основной целевой аудиторией продукта являются:

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		72

- телекоммуникационные компании;
- центры обработки данных;
- разработчики сетевого оборудования;
- предприятия промышленного интернета вещей (IIoT);
- организации, использующие MQTT-инфраструктуры;
- компании, предоставляющие облачные и мультимедийные сервисы;
- научно-исследовательские организации;
- образовательные учреждения технического профиля.

Стратегия выхода на рынок предполагает поэтапное внедрение технологии.

На первом этапе основное внимание планируется сосредоточить на образовательном и исследовательском сегментах. Предполагается внедрение программного комплекса в учебный процесс технических университетов и исследовательских лабораторий, занимающихся сетевыми технологиями, анализом трафика и распределёнными системами. Такой подход позволит провести апробацию технологии в реальных условиях эксплуатации, получить обратную связь от специалистов и сформировать базу практических кейсов применения.

Дополнительно планируется использование программных сетевых симуляторов и тестовых стендов для демонстрации эффективности разработанного решения в условиях высоконагруженных сетей и нестабильных каналов связи. Особое внимание будет уделено моделированию сетевой активности в GNS3 и аналогичных средах с целью подтверждения эффективности методов темпорально-корреляционного анализа при снижении jitter и повышении стабильности QoS/QoE.

На следующем этапе предполагается расширение присутствия проекта в корпоративном сегменте. Основными направлениями внедрения станут:

- IIoT-инфраструктуры;
- промышленные сети;
- облачные платформы;

- edge computing;
- мультимедийные сервисы реального времени;
- корпоративные сети с повышенными требованиями к QoS.

Для потенциальных заказчиков планируется разработка индивидуальных коммерческих предложений с учётом специфики сетевой инфраструктуры и требований к производительности. Особое внимание будет уделено демонстрации практической эффективности продукта, включая:

- снижение вариации задержек;
- повышение стабильности передачи данных;
- оптимизацию обработки пакетных очередей;
- повышение качества пользовательского опыта.

Дополнительно рассматривается возможность сотрудничества с разработчиками отечественного сетевого оборудования и Linux-based платформ для интеграции разработанного блока анализа трафика непосредственно в программные маршрутизаторы и системы packet processing.

3.7 Описание рисков

В таблице 4 представлена оценка потенциальных рисков.

Таблица 4 Оценка рисков проекта

Категория риска	Риск	Вероятность	Влияние	Меры по минимизации
Технологическое	Недостижение заявленных технических параметров (задержка, потребление памяти).	Средняя	Высокое	Поэтапное тестирование, привлечение экспертов (научный руководитель)

Категория риска	Риск	Вероятность	Влияние	Меры по минимизации
Рыночные	Низкая осведомленность рынка, ведущая к долгим циклам продаж в B2B.	Высокая	Среднее	Проведение пилотных внедрений, сотрудничество с образовательными и исследовательскими организациями, публикация результатов тестирования и демонстрация эффективности технологии на практических кейсах.
Финансовые	Недостаток финансирования для завершения разработки и вывода на рынок.	Высокая	Высокое	Поэтапная разработка и частичное внедрение в виде ПО для уже имеющейся аппаратной части.
Кадровые	Нехватка квалифицированных специалистов в регионе.	Средняя	Среднее	Проведение обучения и стажировок для быстрой адаптации команды.

Заключение

В ходе работы создан сетевой маршрутизатор с динамически загружаемыми планировщиками очереди. Реализована многопоточная архитектура с отдельными потоками приёма и отправки для дуплексных направлений, плагиновая система загрузки алгоритмов, захват и отправка сырых IP-пакетов через RAW-сокеты, фильтрация трафика на уровне приложения. Тестирование в среде GNS3 с генерацией трафика утилитой iperf3 подтвердило работоспособность продукта.

В ходе работы реализованы и исследованы четыре алгоритма управления очередью: FIFO, RED, CoDel и ARIMA-pacing. Экспериментальные результаты показали, что ARIMA-pacing обеспечивает наилучший показатель джиттера — 0,318 мс, что на 25,5 % ниже FIFO и на 10,4 % ниже CoDel, при сохранении пропускной способности 49,9 Мбит/с и отсутствии потерь. Это подтверждает эффективность применения прогнозирования временных рядов в задачах управления очередью.

В процессе написания работы изучены RAW-сокеты Linux, механизмы динамической загрузки библиотек, методы оценки коэффициентов авторегрессии с использованием GSL, а также методики тестирования в эмулированной сети GNS3.

Дальнейшее развитие проекта возможно в направлении интеграции с технологиями DPDK и XDP для повышения пропускной способности, добавления новых алгоритмов очереди в виде плагинов, а также создания веб-интерфейса для мониторинга состояния очереди в реальном времени

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		76

Список использованных источников

1. Шоттс У. Командная строка Linux. Полное руководство. 2-е межд. изд. — СПб.: Питер, 2020. — 544 с.: ил. — (Серия «Для профессионалов»).
2. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы. — 5-е изд. — СПб. : Питер, 2016. — 992 с. — (Серия «Учебник для вузов»).
3. Библия Linux. 10-е издание. / Негус Кристофер; перевод с английского языка Павлов А. — СПб.: Питер, 2022. — 928 с.: ил. — (Серия «Для профессионалов»).
4. Таненбаум Э., Уэзеролл Д. Компьютерные сети. — 5-е изд. — СПб. : Питер, 2012. — 912 с. — (Серия «Классика computer science»).
5. Столлингс В. Современные компьютерные сети. — 2-е изд. — СПб. : Питер, 2004. — 783 с.
6. Бокс Дж., Дженкинс Г. Анализ временных рядов. Прогноз и управление. — М. : Мир, 1974. — 406 с.
7. DeepSeek (модель DeepSeek-V3) [Электронный ресурс] / разработчик DeepSeek. — URL: deepseek.com
8. Официальная документация GNU Scientific Library [Электронный ресурс]. — URL: <https://www.gnu.org/software/gsl/doc/> (дата обращения: 20.05.2026).
9. Nichols K., Jacobson V. Controlling Queue Delay // Communications of the ACM. — 2012. — Vol. 55, № 7. — P. 42–50.
10. Floyd S., Jacobson V. Random Early Detection Gateways for Congestion Avoidance // IEEE/ACM Transactions on Networking. — 1993. — Vol. 1, № 4. — P. 397–413.

Приложение А

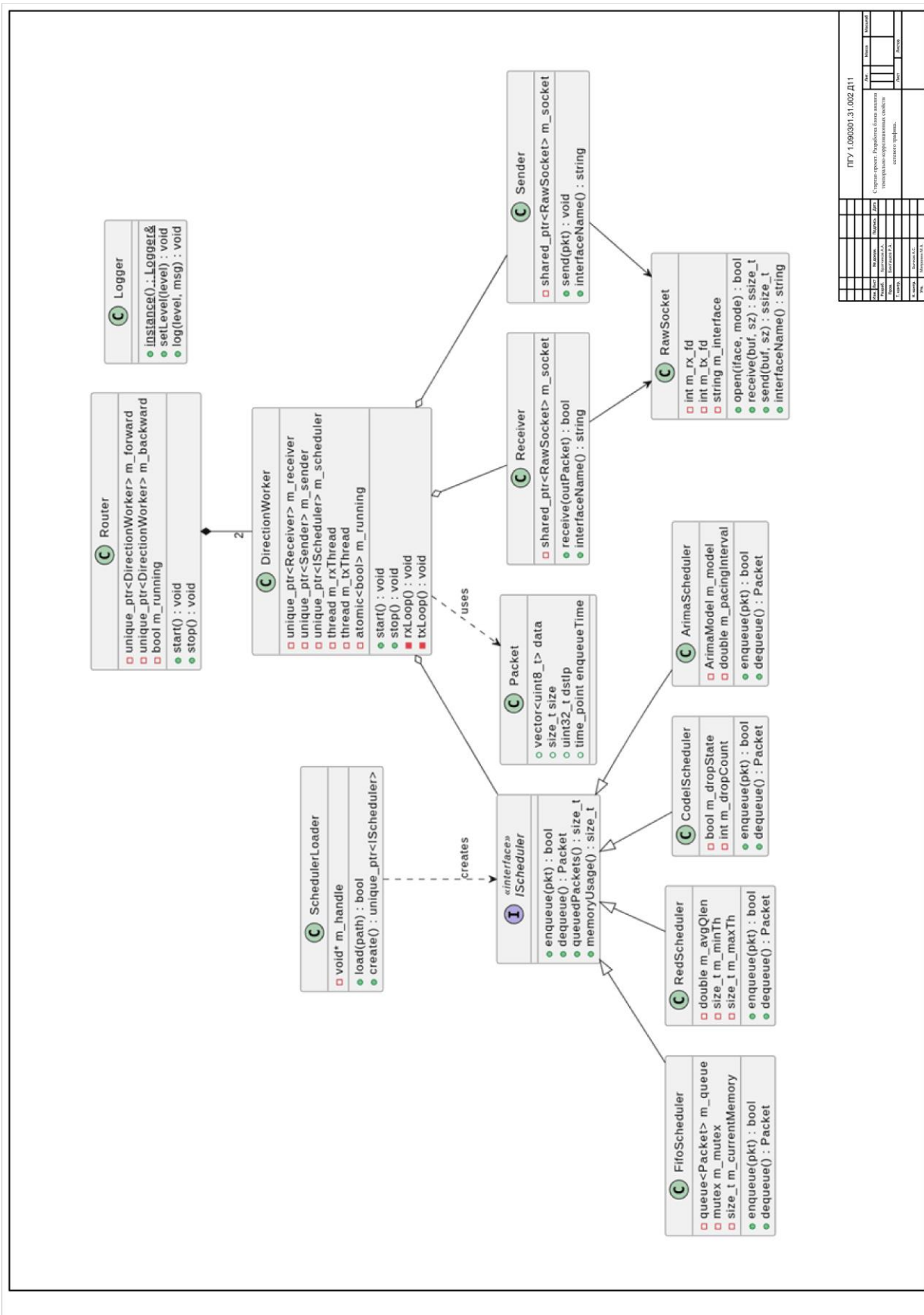
(обязательное)

Стартап-проект. Разработка блока анализа темпорально-корреляционных свойств
сетевого трафика.

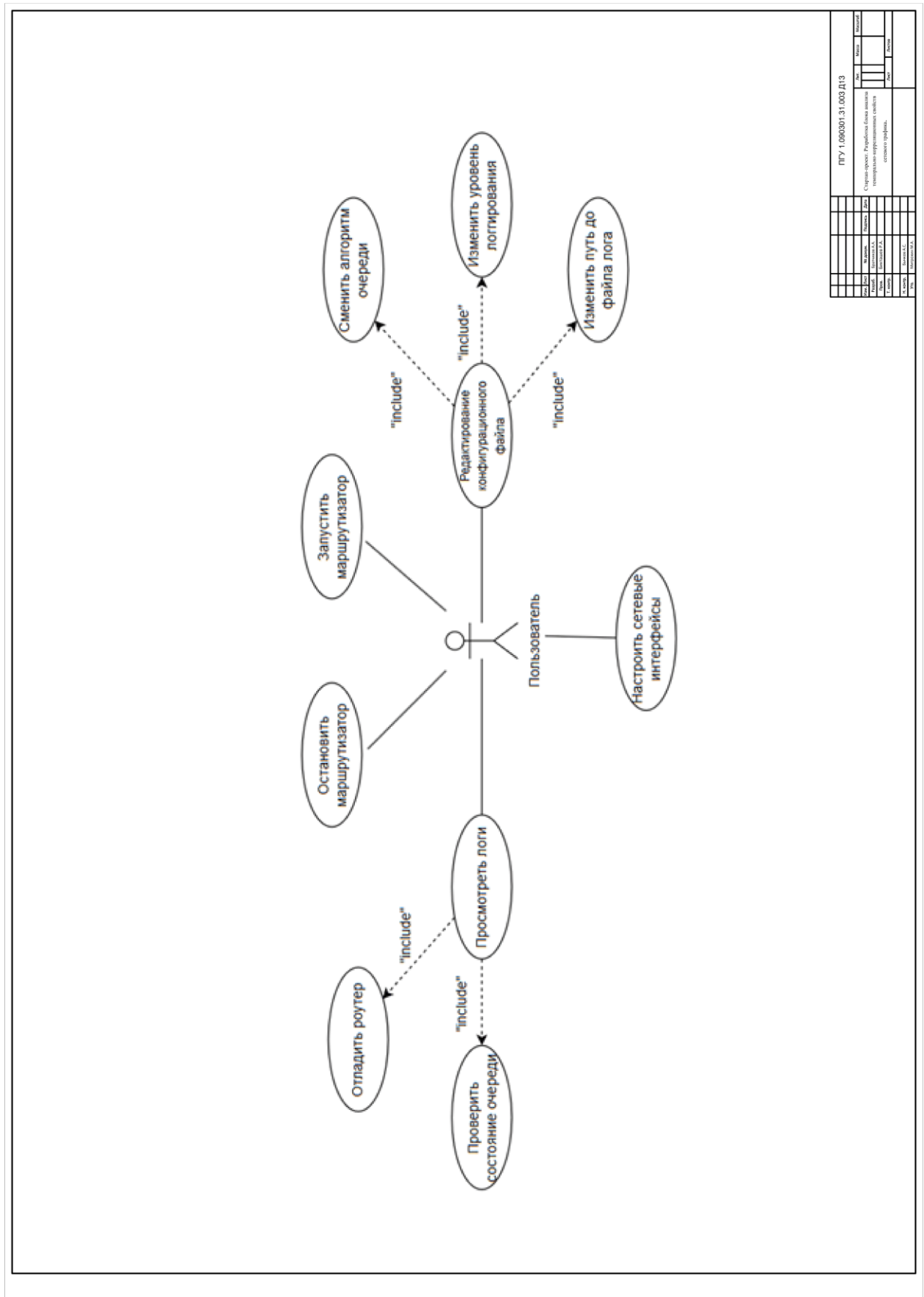
UML-диаграммы

					ПГУ1.090301.31.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

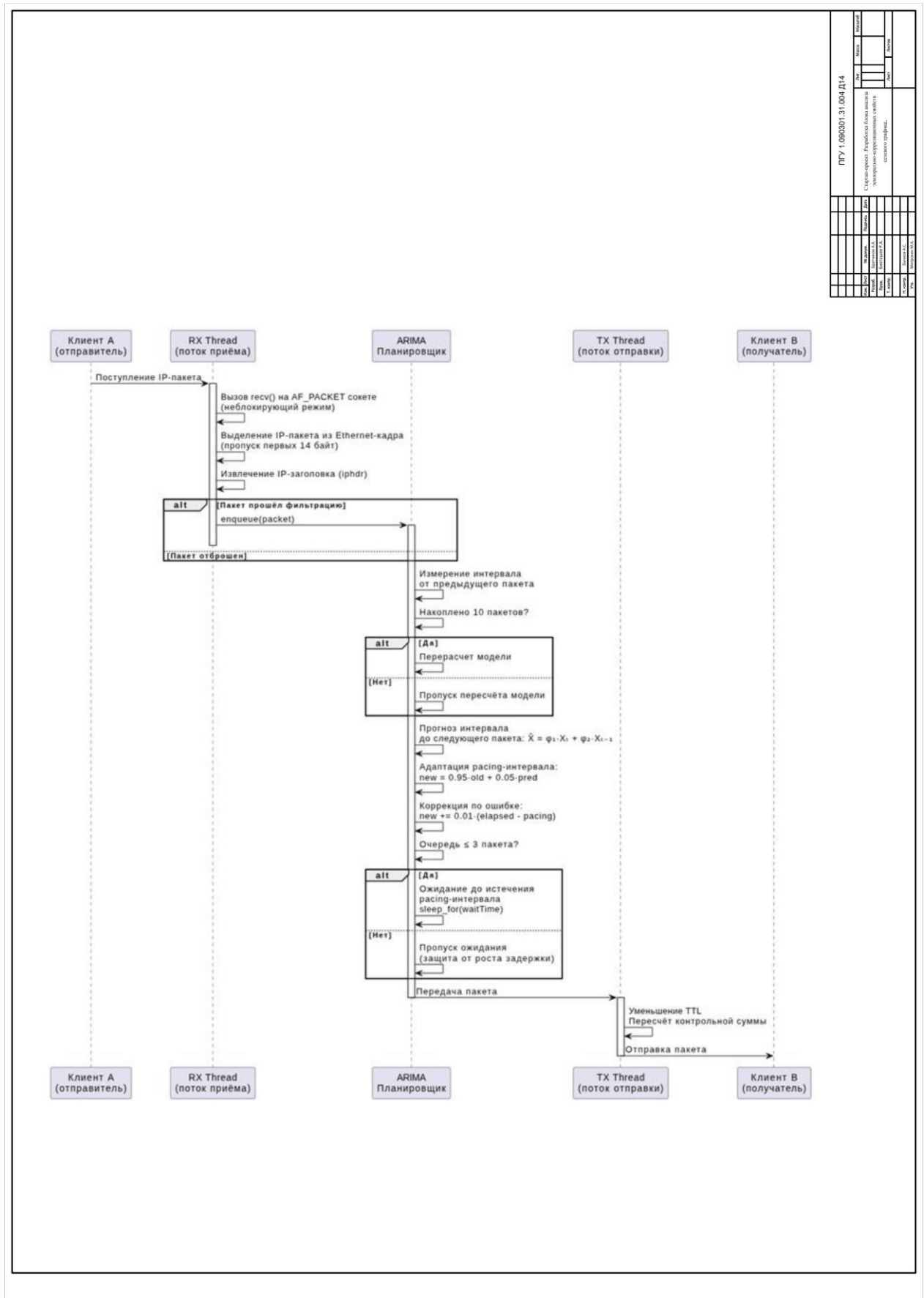
Приложение А.1 – UML-диаграмма классов



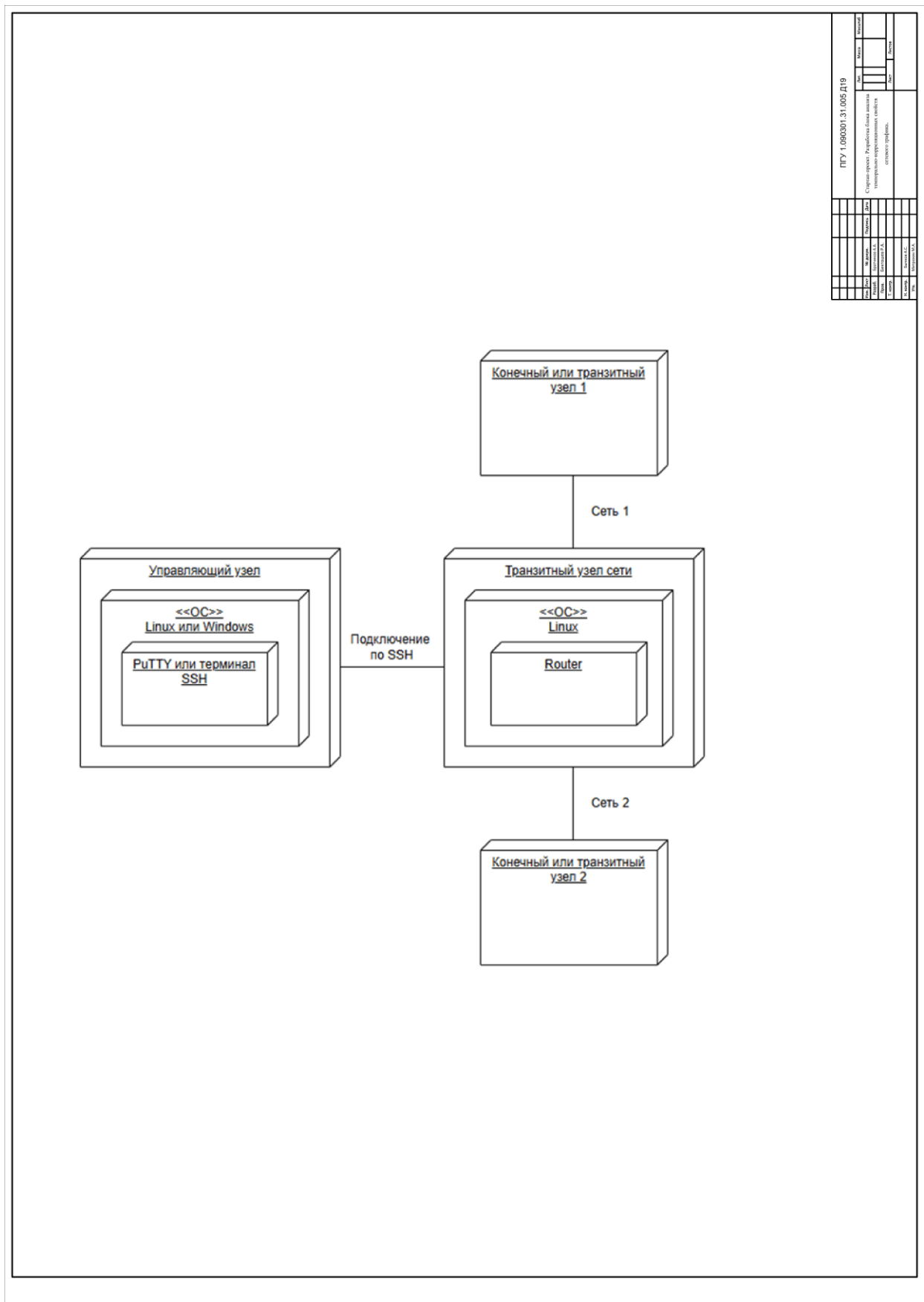
Приложение А.2 – UML-диаграмма прецедентов



Приложение А.3 – UML-диаграмма последовательностей



Приложение А.4 – UML-диаграмма развёртывания

[illegible]

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(обязательное)

Стартап-проект. Разработка блока анализа темпорально-корреляционных свойств сетевого трафика.

Листинг программы

Файл Packet.hpp

```
#pragma once

#include <vector>
#include <cstdint>
#include <chrono>

class Packet
{
public:
    using Clock = std::chrono::steady_clock;
public:
    Packet() = default;
    Packet(const Packet&) = delete;
    Packet& operator=(const Packet&) = delete;
    Packet(Packet&& other) noexcept
        : data(std::move(other.data))
        , size(other.size)
        , dstIp(other.dstIp)
        , enqueueTime(other.enqueueTime)
    {
        other.size = 0;
        other.dstIp = 0;
    }
    explicit Packet(const std::vector<uint8_t>& rawData);

    explicit Packet(std::vector<uint8_t>&& rawData);
```

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		83

```

Packet& operator=(Packet&& other) noexcept
{
    data = std::move(other.data);
    size = other.size;
    dstIp = other.dstIp;
    enqueueTime = other.enqueueTime;
    return this;
}

public:
    std::vector<uint8_t> data;

    size_t size;
    uint32_t dstIp;
    Clock::time_point enqueueTime;
};

```

Файл Packet.cpp

```

#include "Packet.hpp"

Packet::Packet(const std::vector<uint8_t>& rawData)
    : data(rawData),
      size(rawData.size()),
      dstIp(0),
      enqueueTime(Clock::now())
{
}

Packet::Packet(std::vector<uint8_t>&& rawData)
    : data(std::move(rawData)),
      size(data.size()),
      dstIp(0),
      enqueueTime(Clock::now())
{
}

```

```
}
```

Файл RawSocket.hpp

```
#pragma once
#include "utils/Logger.hpp"
#include <cstdint>
#include <cstdint>
#include <cstdint>
#include <cstring>
#include <string>
class RawSocket
{
public:
    enum class Mode
    {
        RX,    // AF_PACKET – приём
        TX     // AF_INET + IPPROTO_RAW – отправка
    };

public:
    RawSocket();
    ~RawSocket();

    bool open(const std::string& interfaceName, Mode mode);

    ssize_t receive(uint8_t buffer, size_t size);
    ssize_t send(const uint8_t buffer, size_t size);
    std::string interfaceName() const { return m_interface; }
private:
    std::string m_interface;
    int m_rx_fd = -1;
    int m_tx_fd = -1;
    Mode m_mode;
};
```

Файл RawSocket.cpp

```
#include "RawSocket.hpp"
```

```

#include <sys/socket.h>
#include <netinet/ip.h>
#include <netpacket/packet.h>
#include <net/ethernet.h>
#include <net/if.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>
#include <cstring>
#include <cerrno>

RawSocket::RawSocket()
{
}

RawSocket::~~RawSocket()
{
    if (m_rx_fd > 0)
        close(m_rx_fd);
    if (m_tx_fd > 0)
        close(m_tx_fd);
}

bool RawSocket::open(const std::string& interfaceName, Mode
mode)
{
    m_mode = mode;
    m_interface = interfaceName;
    if (mode == Mode::RX)
    {
        // Приём: AF_PACKET + ETH_P_ALL (получаем Ethernet-
кадры)

```

```

        m_rx_fd = ::socket(AF_PACKET, SOCK_RAW,
hton(EHT_P_ALL));

        if (m_rx_fd < 0)
        {
            LOG_ERROR("socket(AF_PACKET) failed: " +
std::string(strerror(errno)));
            return false;
        }

        if (!interfaceName.empty())
        {

            struct sockaddr_ll sll;
            memset(&sll, 0, sizeof(sll));
            sll.sll_family = AF_PACKET;
            sll.sll_ifindex =
if_nametoindex(interfaceName.c_str());

            if (sll.sll_ifindex == 0)
            {
                LOG_ERROR("if_nametoindex failed: " +
std::string(strerror(errno)));
                close(m_rx_fd);
                m_rx_fd = -1;
                return false;
            }

            sll.sll_protocol = hton(EHT_P_ALL);

            if (bind(m_rx_fd, (struct sockaddr)&sll,
sizeof(sll)) < 0)
            {

```

```

        LOG_ERROR("bind() failed: " +
std::string(strerror(errno)));
        close(m_rx_fd);
        m_rx_fd = -1;
        return false;
    }
}

fcntl(m_rx_fd, F_SETFL, O_NONBLOCK);
LOG_DEBUG("RX socket fd=" + std::to_string(m_rx_fd) +
" on " + interfaceName);
}
else // Mode::TX
{
    // Отправка: AF_INET + IPPROTO_RAW (ядро добавляет
Ethernet-заголовок)
    m_tx_fd = ::socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

    if (m_tx_fd < 0)
    {
        LOG_ERROR("socket(AF_INET) failed: " +
std::string(strerror(errno)));
        return false;
    }

    if (!interfaceName.empty())
    {
        int one = 1;
        if (setsockopt(m_tx_fd,
                        IPPROTO_IP,
                        IP_HDRINCL,
                        &one,
                        sizeof(one)) < 0)
        {

```

```

        LOG_ERROR("IP_HDRINCL failed: " +
std::string(strerror(errno)));
        close(m_tx_fd);
        m_tx_fd = -1;
        return false;
    }
    if (setsockopt(m_tx_fd,
        SOL_SOCKET,
        SO_BINDTODEVICE,
        interfaceName.c_str(),
        interfaceName.size()) < 0)
    {
        LOG_ERROR("setsockopt() failed: " +
std::string(strerror(errno)));
        close(m_tx_fd);
        m_tx_fd = -1;
        return false;
    }

    }

    fcntl(m_tx_fd, F_SETFL, O_NONBLOCK);
    LOG_DEBUG("TX socket fd=" + std::to_string(m_tx_fd) +
" on " + interfaceName);
}

return true;
}

ssize_t RawSocket::receive(uint8_t buffer, size_t size)
{
    int fd = (m_mode == Mode::RX) ? m_rx_fd : m_tx_fd;

```

```

        ssize_t n = ::recv(fd, buffer, size, 0);

        if (n < 0)
        {
            if (errno == EAGAIN || errno == EWOULDBLOCK) return 0;
            if (errno == EINTR) return 0;
            return -1;
        }

        return n;
    }

    ssize_t RawSocket::send(const uint8_t buffer, size_t size)
    {
        int fd = (m_mode == Mode::RX) ? m_rx_fd : m_tx_fd;

        if (m_mode == Mode::TX)
        {
            auto iph = (iphdr)buffer;

            sockaddr_in dst{};
            dst.sin_family = AF_INET;
            dst.sin_addr.s_addr = iph->daddr;

            ssize_t n = ::sendto(
                fd,
                buffer,
                size,
                0,
                (sockaddr)&dst,
                sizeof(dst)
            );

            if (n < 0)

```

```

        {
            LOG_ERROR("sendto failed: " +
std::string(strerror(errno)));
        }

        return n;
    }

    ssize_t n = ::send(fd, buffer, size, 0);

    if (n < 0)
    {
        if (errno == EAGAIN || errno == EWOULDBLOCK) return 0;
        if (errno == EINTR) return 0;
        return -1;
    }

    return n;
}

```

Файл Receiver.hpp

```

#pragma once
#include "RawSocket.hpp"
#include "core/Packet.hpp"
#include <memory>
#include <cstring>
#include "utils/Logger.hpp"

class Receiver
{
public:
    explicit Receiver(std::shared_ptr<RawSocket> socket);

    ~Receiver();
}

```

```

public:
    // Пытается принять пакет.
    // true -> пакет получен и записан в outPacket
    // false -> ничего не пришло или ошибка
    bool receive(Packet& outPacket);
    std::string interfaceName() const { return m_socket-
>interfaceName(); }
private:
    std::shared_ptr<RawSocket> m_socket;
};

```

Файл Receiver.cpp

```

#include "Receiver.hpp"
#include <vector>
#include <cstdint>
#include <utility>
#include <netinet/ip.h>
#include <net/ethernet.h>
#pragma message "Receiver includes Packet from: " __FILE__
Receiver::Receiver(std::shared_ptr<RawSocket> socket)
    : m_socket(std::move(socket))
{
}

Receiver::~Receiver() = default;

bool Receiver::receive(Packet& outPacket)
{
    constexpr size_t BUFFER_SIZE = 65536;

    std::vector<uint8_t> buffer(BUFFER_SIZE);

    // читаем сырой пакет из ядра

```

```

        ssize_t n = m_socket->receive(buffer.data(),
buffer.size());

        if (n < 0)
        {
            //LOG_ERROR("RawSocket receive error: " +
std::string(strerror(errno)));
            return false; // ничего не пришло или неполный
заголовок
        }
        if (n == 0) {
            // EAGAIN – просто нет данных, это нормально
            return false;
        }
        // Минимум: ethhdr (14) + iphdr (20) = 34 байта
        if (static_cast<size_t>(n) <= sizeof(struct ethhdr) +
sizeof(struct iphdr))
        {
            LOG_DEBUG("Packet too small: " + std::to_string(n) + "
bytes");
            return false;
        }
        LOG_DEBUG("Received " + std::to_string(n) + " bytes");
        LOG_DEBUG("First bytes: " +
            std::to_string(buffer[0]) + " " +
            std::to_string(buffer[1]) + " " +
            std::to_string(buffer[12]) + " " +
            std::to_string(buffer[13]));
        // обрезаем буфер под реальный размер
        buffer.resize(static_cast<size_t>(n));
        std::vector<uint8_t> ipPacket(
            buffer.begin() + sizeof(struct ethhdr),
            buffer.end()
        );

```

```

        auto ip = reinterpret_cast<struct iphdr>(ipPacket.data());
        // Drop broadcast и 0.0.0.0
        if (ip->saddr == 0 || ip->daddr ==
htonl(INADDR_BROADCAST)) {
            return false;
        }

        // Drop не IPv4
        if (ip->version != 4) {
            return false;
        }
        // упаковываем в внутренний Packet
        outPacket = Packet(std::move(ipPacket));

        return true;
    }

```

Файл Sender.hpp

```

#pragma once

#include "core/Packet.hpp"
#include "network/RawSocket.hpp"
#include <memory>

class Sender
{
public:
    explicit Sender(std::shared_ptr<RawSocket> socket);
    ~Sender();

    std::string interfaceName() const { return m_socket-
>interfaceName(); }

    void send(Packet&& pkt);

private:
    std::shared_ptr<RawSocket> m_socket;

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.31.001 ПЗ

Лист

94

```
};
```

Файл Sender.cpp

```
#include "Sender.hpp"
```

```
#include "RawSocket.hpp"
```

```
#include "core/Packet.hpp"
```

```
#include <netinet/ip.h>
```

```
#include <cstring>
```

```
Sender::Sender(std::shared_ptr<RawSocket> socket)
```

```
    : m_socket(std::move(socket))
```

```
{
```

```
}
```

```
Sender::~Sender() = default;
```

```
uint16_t checksum(const void data, size_t len)
```

```
{
```

```
    auto ptr = static_cast<const uint16_t>(data);
```

```
    uint32_t sum = 0;
```

```
    while (len > 1)
```

```
    {
```

```
        sum += ptr++;
```

```
        len -= 2;
```

```
    }
```

```
    if (len)
```

```
        sum += (const uint8_t)ptr;
```

```
    while (sum >> 16)
```

```
        sum = (sum & 0xFFFF) + (sum >> 16);
```

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		95

```

        return static_cast<uint16_t>(~sum);
    }

    void Sender::send(Packet&& pkt)
    {
        if (pkt.data.empty())
            return;

        // смотрим на IP header прямо в буфере
        auto ip = reinterpret_cast<iphdr>(pkt.data.data());

        if (ip->ttl <= 1)
        {
            // TTL истёк — пакет не отправляем
            return;
        }

        ip->ttl--;
        LOG_DEBUG "[" + m_socket->interfaceName() + "] Sending:
TTL=" + std::to_string(ip->ttl) +
            " src=" + ip_to_string(ip->saddr) + " dst=" +
ip_to_string(ip->daddr));

        // checksum нужно пересчитать после изменения TTL
        ip->check = 0;
        ip->check = checksum(reinterpret_cast<uint16_t>(ip),
sizeof(iphdr));

        // отправляем как есть (полный IP пакет уже внутри
pkt.data)
        m_socket->send(pkt.data.data(), pkt.size);
    }

```

Файл IScheduler.hpp

```
#pragma once
```

```
#include <optional>
```

```
#include <cstdint>
```

```
#include "core/Packet.hpp"
```

```
class IScheduler
```

```
{
```

```
public:
```

```
    virtual ~IScheduler() = default;
```

```
public:
```

```
    // Добавить пакет в очередь
```

```
    virtual bool enqueue(Packet&& packet) = 0;
```

```
    // Забрать пакет из очереди
```

```
    virtual Packet dequeue() = 0;
```

```
public:
```

```
    // Количество пакетов в очереди
```

```
    virtual size_t queuedPackets() const = 0;
```

```
    // Использование памяти очередью
```

```
    virtual size_t memoryUsage() const = 0;
```

```
};
```

Файл SchedulerLoader.hpp

```
#pragma once
```

```
#include <memory>
```

```
#include <string>
```

```
class IScheduler;
```

```

class SchedulerLoader
{
public:
    SchedulerLoader();

    ~SchedulerLoader();

public:
    bool load(const std::string& path);

    std::unique_ptr<IScheduler> create();

private:
    using CreateSchedulerFn = IScheduler () ();

private:
    void m_libraryHandle;

    CreateSchedulerFn m_createScheduler;
};

```

Файл SchedulerLoader.cpp

```

#include "SchedulerLoader.hpp"

#include "IScheduler.hpp"

#include <dlfcn.h>
#include <iostream>

SchedulerLoader::SchedulerLoader()
    : m_libraryHandle(nullptr),
      m_createScheduler(nullptr)
{
}

```

```

SchedulerLoader::~SchedulerLoader()
{
    if (m_libraryHandle)
    {
        dlclose(m_libraryHandle);
    }
}

bool SchedulerLoader::load(const std::string& path)
{
    m_libraryHandle = dlopen(path.c_str(), RTLD_NOW);

    if (!m_libraryHandle)
    {
        std::cerr << "dlopen failed: " << dlerror() <<
std::endl;
        return false;
    }

    m_createScheduler = reinterpret_cast<CreateSchedulerFn>(
        dlsym(m_libraryHandle, "createScheduler")
    );

    if (!m_createScheduler)
    {
        std::cerr << "dlsym failed: " << dlerror() <<
std::endl;

        dlclose(m_libraryHandle);
        m_libraryHandle = nullptr;

        return false;
    }
}

```

```

        return true;
    }

std::unique_ptr<IScheduler> SchedulerLoader::create()
{
    if (!m_createScheduler)
    {
        return nullptr;
    }

    return std::unique_ptr<IScheduler>(
        m_createScheduler()
    );
}

```

Файл DirectionWorker.hpp

```

#pragma once
#include <memory>
#include <thread>
#include <atomic>
#include "network/Receiver.hpp"
#include "network/Sender.hpp"
#include "scheduler/IScheduler.hpp"

class DirectionWorker
{
public:
    DirectionWorker(
        std::unique_ptr<Receiver> receiver,
        std::unique_ptr<Sender> sender,
        std::unique_ptr<IScheduler> scheduler
    );

    ~DirectionWorker();
}

```

```

public:
    void start();

    void stop();

private:
    void rxLoop();

    void txLoop();

private:
    std::unique_ptr<Receiver> m_receiver;

    std::unique_ptr<Sender> m_sender;

    std::unique_ptr<IScheduler> m_scheduler;

    std::thread m_rxThread;

    std::thread m_txThread;

    std::atomic<bool> m_running;
};

```

Файл DirectionWorker.cpp

```

#include "DirectionWorker.hpp"

#include <utility>

DirectionWorker::DirectionWorker(
    std::unique_ptr<Receiver> receiver,
    std::unique_ptr<Sender> sender,
    std::unique_ptr<IScheduler> scheduler

```

```

)

: m_receiver(std::move(receiver)),
  m_sender(std::move(sender)),
  m_scheduler(std::move(scheduler)),
  m_running(false)

{

}

DirectionWorker::~DirectionWorker()
{
    stop();
}

void DirectionWorker::start()
{
    if (m_running.load())
        return;

    m_running = true;

    // RX thread
    m_rxThread = std::thread([this]()
    {
        rxLoop();
    });

    // TX thread
    m_txThread = std::thread([this]()
    {
        txLoop();
    });
}

```

```

void DirectionWorker::stop()
{
    if (!m_running.load())
        return;

    m_running = false;

    if (m_rxThread.joinable())
        m_rxThread.join();

    if (m_txThread.joinable())
        m_txThread.join();
}

void DirectionWorker::rxLoop()
{
    LOG_DEBUG("rx loop start on " + m_receiver->interfaceName());

    // Receiver должен внутри читать raw socket
    // и отдавать пакеты в scheduler

    while (m_running.load())
    {
        Packet pkt;

        // идея: receiver наполняет Packet
        if (m_receiver->receive(pkt))
        {
            LOG_DEBUG "[" + m_receiver->interfaceName() + "]
packet recv, size=" + std::to_string(pkt.size));
            bool ok = m_scheduler->enqueue(std::move(pkt));
            LOG_DEBUG("enqueue: " + std::string(ok ? "ok" :
"FAILED"));
        }
    }
}

```

```

    }

    }

}

void DirectionWorker::txLoop()
{
    LOG_DEBUG("tx loop start on " + m_sender->interfaceName());

    while (m_running.load())
    {
        auto pktOpt = m_scheduler->dequeue();

        if (pktOpt.data.empty())
        {

            // важно: не жрем CPU

            std::this_thread::sleep_for(std::chrono::microseconds(50));
            continue;
        }

        auto now = std::chrono::steady_clock::now();
        auto queueTime =
std::chrono::duration_cast<std::chrono::microseconds>(
    now - pktOpt.enqueueTime).count();
        LOG_DEBUG "[" + m_sender->interfaceName() + "] packet
send, waited_in_queue=" +
            std::to_string(queueTime) + "us";
        m_sender->send(std::move(pktOpt));
    }
}

Файл Router.hpp

#pragma once

#include <memory>

```

```

#include "core/DirectionWorker.hpp"

class Router
{
public:
    Router(
        std::unique_ptr<DirectionWorker> forwardWorker,
        std::unique_ptr<DirectionWorker> backwardWorker
    );

    ~Router();

public:
    void start();

    void stop();

private:
    std::unique_ptr<DirectionWorker> m_forwardWorker;

    std::unique_ptr<DirectionWorker> m_backwardWorker;

    bool m_running;
};

```

Файл Router.cpp

```

#include "Router.hpp"

#include <utility>

Router::Router(
    std::unique_ptr<DirectionWorker> forwardWorker,
    std::unique_ptr<DirectionWorker> backwardWorker
)

```

```

        : m_forwardWorker(std::move(forwardWorker)),
          m_backwardWorker(std::move(backwardWorker)),
          m_running(false)
    {
    }

Router::~Router()
{
    stop();
}

void Router::start()
{
    if (m_running)
        return;

    m_running = true;

    if (m_forwardWorker)
        m_forwardWorker->start();

    if (m_backwardWorker)
        m_backwardWorker->start();
}

void Router::stop()
{
    if (!m_running)
        return;

    m_running = false;

    if (m_forwardWorker)
        m_forwardWorker->stop();
}

```

```

        if (m_backwardWorker)
            m_backwardWorker->stop();
    }

```

Файл `Logger.hpp`

```
#pragma once
```

```

#include <iostream>
#include <string>
#include <sstream>
#include <chrono>
#include <sys/syscall.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <fstream>
#include <mutex>
#include <iomanip>
enum class LogLevel
{
    DEBUG,
    INFO,
    WARN,
    ERROR
};

```

```

class Logger
{
public:
    static Logger& instance()
    {
        static Logger logger;
        return logger;
    }
}

```

```

void setLevel(LogLevel level) { m_level = level; }

void setLogFile(const std::string& path)
{
    if (!path.empty())
        m_file.open(path, std::ios::app);
}

void log(LogLevel level, const std::string& message)
{
    if (level < m_level) return;

    std::lock_guard<std::mutex> lock(m_mutex);

    std::string levelStr;
    switch (level)
    {
        case LogLevel::DEBUG: levelStr = "DEBUG"; break;
        case LogLevel::INFO: levelStr = "INFO"; break;
        case LogLevel::WARN: levelStr = "WARN"; break;
        case LogLevel::ERROR: levelStr = "ERROR"; break;
    }

    auto now = std::chrono::system_clock::now();
    auto ms =
std::chrono::duration_cast<std::chrono::milliseconds>(
        now.time_since_epoch()) % 1000;
    auto timer =
std::chrono::system_clock::to_time_t(now);
    auto tm = std::localtime(&timer);

    char timeBuf[32];
    strftime(timeBuf, sizeof(timeBuf), "%Y-%m-%d
%H:%M:%S", &tm);

```

```

        std::ostringstream oss;
        oss << "[" << timeBuf << "." << std::setfill('0') <<
std::setw(3) << ms.count() << "]"
        << " [TID:" << syscall(SYS_gettid) << "]"
        << " [" << levelStr << "]" "
        << message;

        std::string line = oss.str();
        std::cout << line << std::endl;

        if (m_file.is_open())
            m_file << line << std::endl;
    }

private:
    Logger() : m_level(LogLevel::INFO) {}
    LogLevel m_level;
    std::ofstream m_file;
    std::mutex m_mutex;
};

// Макросы
#define LOG_DEBUG(msg) Logger::instance().log(LogLevel::DEBUG,
msg)
#define LOG_INFO(msg)  Logger::instance().log(LogLevel::INFO,
msg)
#define LOG_WARN(msg)  Logger::instance().log(LogLevel::WARN,
msg)
#define LOG_ERROR(msg) Logger::instance().log(LogLevel::ERROR,
msg)

static inline std::string ip_to_string(uint32_t ip)
{
    char buf[INET_ADDRSTRLEN];

```

```

        inet_ntop(AF_INET, &ip, buf, sizeof(buf));
        return std::string(buf);
    }
}

Файл Config.hpp

#pragma once

#include <string>
#include <unordered_map>
#include <fstream>
#include <sstream>
#include <algorithm>

class ConfigLoader
{
public:
    bool load(const std::string& path)
    {
        std::ifstream file(path);
        if (!file.is_open()) return false;

        std::string section;
        std::string line;

        while (std::getline(file, line))
        {
            // Убираем пробелы и комментарии
            line = trim(line);
            if (line.empty() || line[0] == '#' || line[0] ==
';')
                continue;

            // Секция [name]
            if (line[0] == '[' && line.back() == ']')
            {

```

```

        section = line.substr(1, line.size() - 2);
        continue;
    }

    // Ключ = значение
    auto pos = line.find('=');
    if (pos != std::string::npos)
    {
        std::string key = trim(line.substr(0, pos));
        std::string value = trim(line.substr(pos +
1));

        m_data[section + "." + key] = value;
    }
}

return true;
}

std::string get(const std::string& section, const
std::string& key,

                const std::string& defaultValue = "")
const
{
    auto it = m_data.find(section + "." + key);
    return (it != m_data.end()) ? it->second :
defaultValue;
}

private:
    std::unordered_map<std::string, std::string> m_data;

    static std::string trim(const std::string& s)
    {
        auto start = s.find_first_not_of(" \t\r\n");

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.31.001 ПЗ

Лист

111

```

        auto end = s.find_last_not_of(" \t\r\n");
        return (start == std::string::npos) ? "" :
s.substr(start, end - start + 1);
    }
};

```

Файл main.cpp

```

#include <iostream>
#include <memory>
#include "utils/Logger.hpp"
#include "core/Router.hpp"
#include "core/DirectionWorker.hpp"
#include "network/RawSocket.hpp"
#include "network/Sender.hpp"
#include "network/Receiver.hpp"
#include "scheduler/SchedulerLoader.hpp"
#include "utils/Config.hpp"

int main()
{
    try
    {
        ConfigLoader config;
        if (!config.load("config.ini"))
        {
            LOG_ERROR("Failed to load config.ini, using
defaults");
        }
        std::cout << "Starting router..." << std::endl;
        std::string logLevel = config.get("logging", "level",
"INFO");

        switch (logLevel[0])    // первый символ
        {
            case 'D':
                Logger::instance().setLevel(LogLevel::DEBUG); break;

```

```

        case 'I':
Logger::instance().setLevel(LogLevel::INFO); break;
        case 'W':
Logger::instance().setLevel(LogLevel::WARN); break;
        case 'E':
Logger::instance().setLevel(LogLevel::ERROR); break;
    }
    std::string logFile = config.get("logging", "logfile",
"");
    Logger::instance().setLogFile(logFile);

    // Интерфейсы из конфига
    std::string ifaceA = config.get("interfaces",
"iface_a", "ens36");
    std::string ifaceB = config.get("interfaces",
"iface_b", "ens37");

    LOG_INFO("Interface A: " + ifaceA);
    LOG_INFO("Interface B: " + ifaceB);

    // Плагин из конфига
    std::string pluginPath = config.get("scheduler",
"plugin", "./plugins/libquery_algorithm.so");
    LOG_INFO("Loading scheduler plugin: " + pluginPath);

    // Scheduler loading (.so)
    SchedulerLoader loader;

    if (!loader.load(pluginPath))
    {
        LOG_ERROR("Failed to load scheduler library");
        return 1;
    }

    auto rxSocketA = std::make_shared<RawSocket>();

```

```

rxSocketA->open(ifaceA, RawSocket::Mode::RX);

// Передатчик – TX сокет
auto txSocketB = std::make_shared<RawSocket>();
txSocketB->open(ifaceB, RawSocket::Mode::TX);

auto forwardWorker =
std::make_unique<DirectionWorker>(
    std::make_unique<Receiver>(rxSocketA),
    std::make_unique<Sender>(txSocketB),
    loader.create()
);

// Аналогично для обратного направления
auto rxSocketB = std::make_shared<RawSocket>();
rxSocketB->open(ifaceB, RawSocket::Mode::RX);

auto txSocketA = std::make_shared<RawSocket>();
txSocketA->open(ifaceA, RawSocket::Mode::TX);

auto backwardWorker =
std::make_unique<DirectionWorker>(
    std::make_unique<Receiver>(rxSocketB),
    std::make_unique<Sender>(txSocketA),
    loader.create()
);

// Router
Router router(
    std::move(forwardWorker),
    std::move(backwardWorker)
);

// Start system

```

```

        router.start();

        LOG_INFO("Router is running. Press ENTER to stop...");
        std::cin.get();

        router.stop();

        LOG_INFO("Router stopped.");
    }
    catch (const std::exception& ex)
    {
        std::cerr << "Fatal error: " << ex.what() <<
std::endl;
        return 1;
    }

    return 0;
}

```

Файл FifoScheduler.cpp

```

#include "scheduler/IScheduler.hpp"
#include "core/Packet.hpp"
#include <queue>
#include <mutex>
#include <condition_variable>
#include "utils/Logger.hpp"
class FifoScheduler : public IScheduler
{
public:
    FifoScheduler()
    {
        m_currentMemory = 0;
        m_maxMemory=12810241024;
    }
}

```

```

~FifoScheduler() override = default;

public:
    bool enqueue(Packet&& packet) override
    {
        std::lock_guard<std::mutex> lock(m_mutex);

        // проверка по памяти
        if (m_currentMemory + packet.size > m_maxMemory)
        {
            return false;
        }

        m_currentMemory += packet.size;

        m_queue.push(std::move(packet));
        if(m_queue.size()>5){
            LOG_INFO("scheduler enqueue: queue size=" +
std::to_string(m_queue.size()));
        }
        m_cv.notify_one();

        return true;
    }

    Packet dequeue() override
    {
        std::unique_lock<std::mutex> lock(m_mutex);
        LOG_DEBUG("scheduler dequeue: waiting, queue size=" +
std::to_string(m_queue.size()));
        m_cv.wait(lock, [&]
        {
            return !m_queue.empty();
        });
    }

```

```

        Packet pkt = std::move(m_queue.front());
        m_queue.pop();

        m_currentMemory -= pkt.size;
        LOG_DEBUG("scheduler dequeue: woke up");
        return pkt;
    }

public:
    size_t queuedPackets() const override
    {
        std::lock_guard<std::mutex> lock(m_mutex);
        return m_queue.size();
    }

    size_t memoryUsage() const override
    {
        std::lock_guard<std::mutex> lock(m_mutex);
        return m_currentMemory;
    }

private:
    mutable std::mutex m_mutex;
    std::condition_variable m_cv;

    std::queue<Packet> m_queue;

    size_t m_currentMemory;
    size_t m_maxMemory;
};

extern "C" {

    // Функция-конструктор планировщика

```

```

IScheduler createScheduler()
{
    // Создаём новый объект FifoScheduler с максимальным
размером памяти по умолчанию (128 МВ)
    return new FifoScheduler();
}
}

```

Файл RedScheduler.cpp

```

#include "scheduler/IScheduler.hpp"
#include "core/Packet.hpp"
#include <queue>
#include <mutex>
#include <condition_variable>
#include <cstdlib>
#include <cmath>
#include "utils/Logger.hpp"

class RedScheduler : public IScheduler
{
public:
    RedScheduler()
        : m_currentMemory(0)
        , m_avgQlen(0.0)
    {}

    bool enqueue(Packet&& packet) override
    {
        std::lock_guard<std::mutex> lock(m_mutex);

        if (m_currentMemory + packet.size > m_maxMemory)
            return false;

        size_t qlen = m_queue.size() 1500;
        m_avgQlen = (1.0 - m_wq) m_avgQlen + m_wq qlen;
    }
}

```

```

        if (m_avgQlen > m_maxTh)
            return false;

        if (m_avgQlen > m_minTh)
        {
            double Pb = m_maxP * (m_avgQlen - m_minTh) /
(m_maxTh - m_minTh);
            if ((double)rand() / RAND_MAX < Pb)
                return false;
        }

        m_currentMemory += packet.size;
        m_queue.push(std::move(packet));
        if(m_queue.size()>5) {
            LOG_INFO("scheduler enqueue: queue size=" +
std::to_string(m_queue.size()));
        }
        m_cv.notify_one();
        return true;
    }

Packet dequeue() override
{
    std::unique_lock<std::mutex> lock(m_mutex);
    m_cv.wait(lock, [&]{ return !m_queue.empty(); });
    Packet pkt = std::move(m_queue.front());
    m_queue.pop();
    m_currentMemory -= pkt.size;
    return pkt;
}

size_t queuedPackets() const override
{

```

```

        std::lock_guard<std::mutex> lock(m_mutex);
        return m_queue.size();
    }

    size_t memoryUsage() const override
    {
        std::lock_guard<std::mutex> lock(m_mutex);
        return m_currentMemory;
    }

private:
    mutable std::mutex m_mutex;
    std::condition_variable m_cv;
    std::queue<Packet> m_queue;

    size_t m_currentMemory;
    double m_avgQlen;

    // Параметры RED
    static constexpr size_t m_maxMemory = 128  1024  1024; //
128 MB
    static constexpr size_t m_minTh = 100  1500;
// 100 пакетов
    static constexpr size_t m_maxTh = 300  1500;
// 300 пакетов
    static constexpr double m_maxP = 0.1;
// 10%
    static constexpr double m_wq = 0.002;
// вес
    };

extern "C" IScheduler createScheduler() {
    return new RedScheduler();
}

```

Файл CodelScheduler.cpp

```
#include "IScheduler.hpp"
#include "Packet.hpp"
#include <queue>
#include <mutex>
#include <condition_variable>
#include <chrono>
#include <cmath>
#include "utils/Logger.hpp"

class CodelScheduler : public IScheduler
{
public:
    CodelScheduler()
        : m_currentMemory(0)
        , m_dropState(false)
        , m_dropCount(0)
        , m_firstDropTime(std::chrono::steady_clock::now())
    {}

    bool enqueue(Packet&& packet) override
    {
        std::lock_guard<std::mutex> lock(m_mutex);

        if (m_currentMemory + packet.size > m_maxMemory)
            return false;

        auto now = std::chrono::steady_clock::now();
        auto sojourn =
std::chrono::duration_cast<std::chrono::microseconds>(
            now - packet.enqueueTime).count();

        if (sojourn < m_targetDelay)
        {
```

```

        m_dropState = false;
        m_dropCount = 0;
    }
    else
    {
        if (!m_dropState)
        {
            m_dropState = true;
            m_firstDropTime = now;
            m_dropCount = 1;
            return false; // дропаем первый
        }

        auto dropInterval = m_interval /
std::sqrt(m_dropCount);

        if (now - m_firstDropTime >
std::chrono::microseconds((long)dropInterval))
        {
            m_firstDropTime = now;
            m_dropCount++;
            return false; // дрон
        }
    }

    m_currentMemory += packet.size;
    m_queue.push(std::move(packet));
    if(m_queue.size()>5){
        LOG_INFO("scheduler enqueue: queue size=" +
std::to_string(m_queue.size()));
    }
    m_cv.notify_one();
    return true;
}

```

```

Packet dequeue() override
{
    std::unique_lock<std::mutex> lock(m_mutex);
    m_cv.wait(lock, [&]{ return !m_queue.empty(); });
    Packet pkt = std::move(m_queue.front());
    m_queue.pop();
    m_currentMemory -= pkt.size;
    return pkt;
}

size_t queuedPackets() const override
{
    std::lock_guard<std::mutex> lock(m_mutex);
    return m_queue.size();
}

size_t memoryUsage() const override
{
    std::lock_guard<std::mutex> lock(m_mutex);
    return m_currentMemory;
}

private:
    mutable std::mutex m_mutex;
    std::condition_variable m_cv;
    std::queue<Packet> m_queue;

    size_t m_currentMemory;
    bool m_dropState;
    int m_dropCount;
    std::chrono::steady_clock::time_point m_firstDropTime;

    // Параметры CoDel

```

```

        static constexpr size_t m_maxMemory = 128  1024  1024;  //
128 MB
        static constexpr long m_targetDelay = 5000;
// 5 ms
        static constexpr long m_interval = 100000;
// 100 ms
    };

```

```

extern "C" IScheduler createScheduler() {
    return new CodeIScheduler();
}

```

Файл ArimaScheduler.cpp

```

#include "IScheduler.hpp"
#include "Packet.hpp"
#include <queue>
#include <mutex>
#include <condition_variable>
#include <chrono>
#include <cmath>
#include <deque>
#include <thread>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_statistics.h>

class ArimaModel
{
public:
    ArimaModel(int p, int d, int q)
        : m_p(p), m_d(d), m_q(q), m_windowSize(100),
m_packetCount(0)
    {
        m_R = gsl_matrix_alloc(p, p);

```

```

        m_r = gsl_vector_alloc(p);
        m_phi = gsl_vector_alloc(p);
        m_perm = gsl_permutation_alloc(p);
    }

~ArimaModel()
{
    gsl_matrix_free(m_R);
    gsl_vector_free(m_r);
    gsl_vector_free(m_phi);
    gsl_permutation_free(m_perm);
}

void addObservation(double value)
{
    m_series.push_back(value);
    if ((int)m_series.size() > m_windowSize)
        m_series.pop_front();
    m_packetCount++;
}

bool shouldReestimate() const
{
    return m_packetCount >= 10 && (int)m_series.size() >=
m_p + 1;
}

void estimate()
{
    if (m_series.size() < m_p + 1) return;
    m_packetCount = 0;

    int n = m_series.size();
    double mean = gsl_stats_mean(&m_series[0], 1, n);

```

```

std::vector<double> autocorr(m_p + 1);
for (int k = 0; k <= m_p; k++)
{
    double cov = 0;
    for (int t = 0; t < n - k; t++)
        cov += (m_series[t] - mean) * (m_series[t + k]
- mean);

    autocorr[k] = cov / n;
}

for (int i = 0; i < m_p; i++)
{
    gsl_vector_set(m_r, i, autocorr[i + 1] /
autocorr[0]);

    for (int j = 0; j < m_p; j++)
        gsl_matrix_set(m_R, i, j, autocorr[abs(i - j)]
/ autocorr[0]);
}

gsl_linalg_LU_decomp(m_R, m_perm, &m_signum);
gsl_linalg_LU_solve(m_R, m_perm, m_r, m_phi);

m_arCoeffs.resize(m_p);
for (int i = 0; i < m_p; i++)
    m_arCoeffs[i] = gsl_vector_get(m_phi, i);
}

double forecast()
{
    if (m_arCoeffs.empty() || m_series.empty()) return 0;
    double pred = 0;
    int n = m_series.size();
    for (int i = 0; i < m_p && i < n; i++)

```

```

        pred += m_arCoeffs[i] * m_series[n - 1 - i];
    return std::max(0.0, pred);
}

bool hasModel() const { return !m_arCoeffs.empty(); }

private:
    int m_p, m_d, m_q, m_windowSize, m_packetCount;
    std::deque<double> m_series;
    std::vector<double> m_arCoeffs;
    gsl_matrix* m_R;
    gsl_vector* m_r;
    gsl_vector* m_phi;
    gsl_permutation* m_perm;
    int m_signum;
};

//
=====

class ArimaScheduler : public IScheduler
{
public:
    ArimaScheduler()
        : m_currentMemory(0)
        , m_model(2, 1, 1)
        , m_lastEnqueue(std::chrono::steady_clock::now())
        , m_lastDequeue(std::chrono::steady_clock::now())
        , m_pacingInterval(1000.0)
        , m_targetJitter(0.0)
        , m_measuredJitter(0.0)
    {}

    bool enqueue(Packet&& packet) override

```

```

{
    std::lock_guard<std::mutex> lock(m_mutex);

    if (m_currentMemory + packet.size > m_maxMemory)
        return false;

    auto now = std::chrono::steady_clock::now();
    double interval =
std::chrono::duration_cast<std::chrono::microseconds>(
        now - m_lastEnqueue).count() / 1000.0;

    if (interval > 0 && interval < 1000)
        m_model.addObservation(interval);

    m_lastEnqueue = now;

    if (m_model.shouldReestimate())
        m_model.estimate();

    m_currentMemory += packet.size;
    m_queue.push(std::move(packet));
    m_cv.notify_one();
    return true;
}

```

Packet dequeue() override

```

{
    std::unique_lock<std::mutex> lock(m_mutex);
    m_cv.wait(lock, [&]{ return !m_queue.empty(); });

    auto now = std::chrono::steady_clock::now();
    auto elapsed =
std::chrono::duration_cast<std::chrono::microseconds>(
        now - m_lastDequeue).count();
}

```

```

// Прогноз ARIMA (медленная адаптация)
if (m_model.hasModel() && m_queue.size() > 3)
{
    double pred = m_model.forecast();
    if (pred > 0)
        m_pacingInterval = 0.95 * m_pacingInterval + 0.05
* pred;
}

// Коррекция по ошибке (очень плавная)
if (m_lastDequeue.time_since_epoch().count() > 0)
{
    double error = elapsed - m_pacingInterval;
    m_pacingInterval += 0.01 * error;
}

m_pacingInterval = std::max(50.0, std::min(2000.0,
m_pacingInterval));

// Pacing ТОЛЬКО если очередь маленькая
if (elapsed < m_pacingInterval && m_queue.size() <= 3)
{
    auto waitTime = std::chrono::microseconds(
        static_cast<int64_t>(m_pacingInterval - elapsed));
    lock.unlock();
    std::this_thread::sleep_for(waitTime);
    lock.lock();
}

Packet pkt = std::move(m_queue.front());
m_queue.pop();
m_currentMemory -= pkt.size;
m_lastDequeue = std::chrono::steady_clock::now();

```

```

        return pkt;
    }

    size_t queuedPackets() const override
    {
        std::lock_guard<std::mutex> lock(m_mutex);
        return m_queue.size();
    }

    size_t memoryUsage() const override
    {
        std::lock_guard<std::mutex> lock(m_mutex);
        return m_currentMemory;
    }

private:
    mutable std::mutex m_mutex;
    std::condition_variable m_cv;
    std::queue<Packet> m_queue;
    size_t m_currentMemory;

    ArimaModel m_model;
    std::chrono::steady_clock::time_point m_lastEnqueue;
    std::chrono::steady_clock::time_point m_lastDequeue;
    double m_pacingInterval;
    double m_targetJitter;
    double m_measuredJitter;

    static constexpr size_t m_maxMemory = 128 * 1024 * 1024;
};

extern "C" IScheduler* createScheduler() {
    return new ArimaScheduler();
}

```

```
}}
```

Файл CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16)

project(RouterProject LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)

option(BUILD_SCHEDULER_PLUGIN "Build scheduler as shared
library" ON)

include_directories(src)

file(GLOB_RECURSE CORE_SOURCES src/core/*.cpp)
file(GLOB_RECURSE NETWORK_SOURCES src/network/*.cpp)
file(GLOB_RECURSE SCHEDULER_SOURCES src/scheduler/*.cpp)
file(GLOB_RECURSE MAIN_SOURCE src/main.cpp)
file(GLOB_RECURSE UTIL_SOURCES src/util/*.cpp)
add_executable(router
    ${MAIN_SOURCE}
    ${CORE_SOURCES}
    ${NETWORK_SOURCES}
    ${SCHEDULER_SOURCES}
    ${UTIL_SOURCES}
)

find_package(Threads REQUIRED)

target_link_libraries(router PRIVATE Threads::Threads dl)

target_compile_options(router PRIVATE
    -Wall
```

					ПГУ1.090301.31.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		131

```

        -Wextra
        -Wpedantic
    )

    if (BUILD_SCHEDULER_PLUGIN)
        set(SCHEDULER_INCLUDES
            src/scheduler
            src/core)

        set(PLUGIN_OUTPUT_DIR ${CMAKE_BINARY_DIR}/plugins)

        add_library(fifo_scheduler SHARED
            plugins/FifoScheduler.cpp)
        target_include_directories(fifo_scheduler PRIVATE
            ${SCHEDULER_INCLUDES})
        set_target_properties(fifo_scheduler PROPERTIES
            LIBRARY_OUTPUT_DIRECTORY ${PLUGIN_OUTPUT_DIR})
        add_library(red_scheduler SHARED plugins/RedScheduler.cpp)
        target_include_directories(red_scheduler PRIVATE
            ${SCHEDULER_INCLUDES})
        set_target_properties(red_scheduler PROPERTIES
            LIBRARY_OUTPUT_DIRECTORY ${PLUGIN_OUTPUT_DIR})
        add_library(codel_scheduler SHARED
            plugins/CodelScheduler.cpp)
        target_include_directories(codel_scheduler PRIVATE
            ${SCHEDULER_INCLUDES})
        set_target_properties(codel_scheduler PROPERTIES
            LIBRARY_OUTPUT_DIRECTORY ${PLUGIN_OUTPUT_DIR})

        add_library(arima_scheduler SHARED
            plugins/ArimaScheduler.cpp)
        target_include_directories(arima_scheduler PRIVATE
            ${SCHEDULER_INCLUDES})

```

```

        set_target_properties(arima_scheduler PROPERTIES
LIBRARY_OUTPUT_DIRECTORY ${PLUGIN_OUTPUT_DIR})
        target_link_libraries(arima_scheduler gsl gslcblas)
    endif()
    option(DIAG_BUILD "Enable deep diagnostics" ON)

    if (DIAG_BUILD)
        set(CMAKE_VERBOSE_MAKEFILE OFF)
        add_compile_options(-v)
        add_compile_options(-H)
        add_compile_options(-fno-elide-constructors)
    endif()

```

					ПГУ1.090301.31.001 ПЗ	Лист
						133
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение С

(обязательное)

Стартап-проект. Разработка блока анализа темпорально-корреляционных
свойств сетевого трафика.

Презентация

					ПГУ1.090301.31.001 ПЗ	Лист
						134
Изм.	Лист	№ докум.	Подпись	Дата		

Стартап-проект. Разработка блока анализа темпорально- корреляционных свойств трафика

Автор работы:

студент гр. 22ВВП2 Братчиков Александр Андреевич

Научный руководитель:

к.т.н., доцент каф. ВТ Бикташев Равиль Айнулович

Пенза, 2026

Цель работы и объекты

Цель работы – разработка и реализация блока анализа темпорально-корреляционных свойств сетевого трафика, обеспечивающего прогнозирование межпакетных интервалов и адаптивное управление очередью для минимизации джиттера.

Объекты работы – программный маршрутизатор и блок анализа темпорально-корреляционных характеристик, выполняющие следующие функции:

- Анализ траффика
- Буферизация с целью уменьшения джиттера
- Маршрутизация траффика

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.31.001 ПЗ

Лист

135

Что такое джиттер?

Джиттер — это вариация или нестабильность времени задержки (latency) при передаче данных между устройствами.

В данной работе рассматривается джиттер последовательности пакетов (джиттер задержки), т.е. разница между задержками двух последовательно прибывших на интерфейс пакетов.

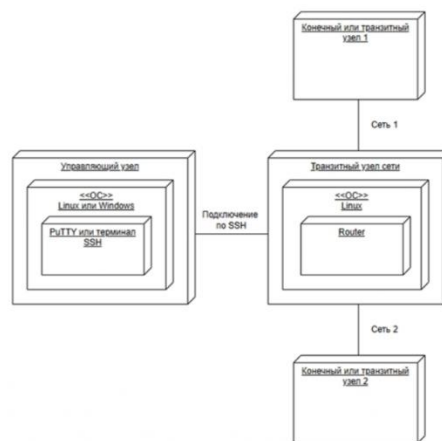
Сравнение аналогов

ПО маршрутизатора	Функциональные возможности	Производительность (джиттер при скорости 500Кб/с)	Сложность использования и развертывания
Cisco IOS / IOS-XE	Широкий набор функций: QoS (CBWFQ, LLQ), ACL, NAT, MPLS, VPN, гибкая маршрутизация, поддержка большого числа протоколов	6.331 мс	Высокая, определяется сложностью командного интерфейса и большим количеством настраиваемых параметров
MikroTik RouterOS	Базовый и расширенный QoS (queues, simple queues, queue tree), firewall, NAT, VPN, routing, удобный GUI (WinBox)	0.071 мс	Низкая обусловлена наличием графических инструментов настройки и относительно простой логикой конфигурации

Используемые технологии



Структура развёртываемого маршрутизатора



Анализ темпорально-корреляционных свойств трафика

Для анализа темпорально-корреляционных свойств трафика используется модель ARIMA(2,1,1), которая включает в себя:

AR(2) — авторегрессия:

Прогноз разности строится по двум предыдущим значениям: $\Delta X_{t+1} = \varphi_1 \cdot \Delta X_t + \varphi_2 \cdot \Delta X_{t-1}$.

Коэффициенты φ_1 , φ_2 находятся методом Юла-Уокера через автокорреляции $\rho(0)$, $\rho(1)$, $\rho(2)$ и LU-разложение матрицы Тейлора.

I(1) — дифференцирование:

Ряд интервалов приводится к стационарному виду взятием разностей: $\Delta X_t = X_t - X_{t-1}$.

Устраняет тренд изменения скорости трафика.

MA(1) — скользящее среднее:

Прогноз корректируется на ошибку предыдущего шага: $\Delta X_{t+1} += \theta_1 \cdot \varepsilon_t$. Коэффициент θ_1 оценивается по дисперсии остатков AR-модели.

При помощи полученных коэффициентов строится прогноз следующего интервала X_t .

Прогноз используется для коррекции интервала выпуска пакетов: $\text{pacing} = 0.95 \cdot \text{pacing} + 0.05 \cdot X_t$.

Дополнительно корректируется по ошибке: $\text{pacing} += 0.01 \cdot (\text{elapsed} - \text{pacing})$. Ограничен диапазоном [50, 2000] мкс

Дальнейшее развитие

Дальнейшее развитие предполагает использование следующих подходов:

FARIMA (Fractional ARIMA):

Учёт долгосрочной памяти трафика через дробный параметр дифференцирования $d \in (0, 0.5)$.

В отличие от ARIMA(2,1,1) с фиксированным $d=1$, FARIMA позволяет моделировать самоподобный характер сетевого трафика, где корреляции сохраняются на больших временных масштабах. Оценка d — нелинейная оптимизация (MLE, wavelet-метод).

Мультифрактальный анализ:

Сетевой трафик обладает мультифрактальной структурой — характеристики масштабирования меняются во времени (всплески, паузы). Метод WTMM (Wavelet Transform Modulus Maxima) позволяет выделить моменты резкого изменения интенсивности и построить спектр сингулярностей $D(h)$ для классификации состояний трафика.

Интеграция с DPDK/XDP:

Перенос обработки в kernel-bypass для скоростей 10+ Гбит/с.

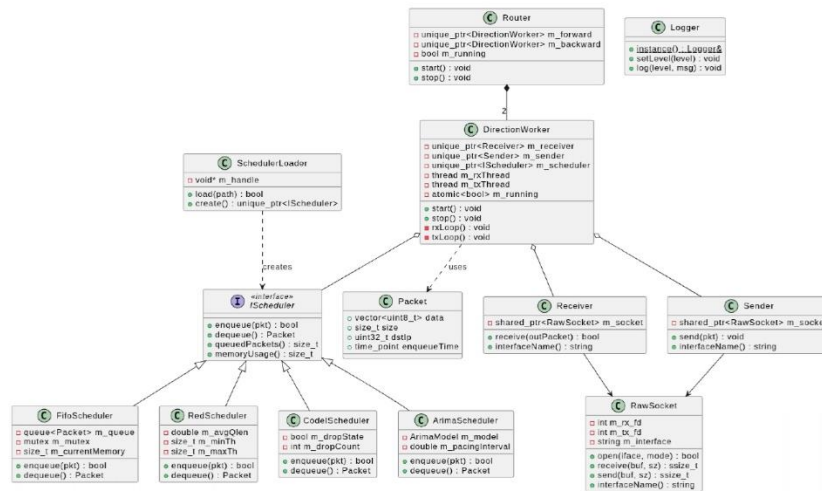
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.31.001 ПЗ

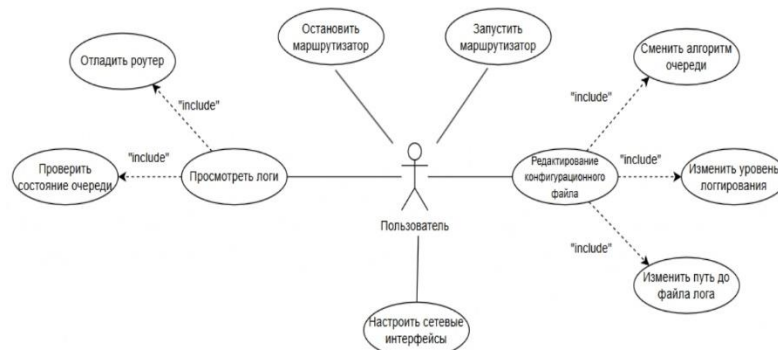
Лист

138

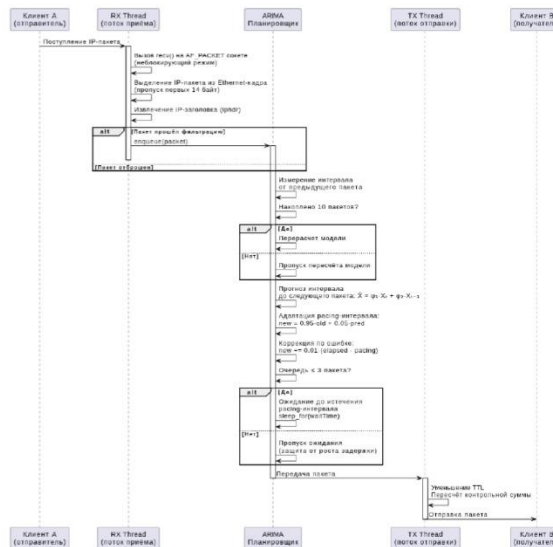
UML-диаграмма классов



UML-диаграмма вариантов использования



UML-диаграмма последовательности



Тестирование FIFO И RED

Server listening on 5201

Accepted connection from 10.0.0.110, port 34610

ID	Interval	Transfer	Bitrate	Jitter	Lost/Total	Delay
1	0.00-1.00	sec 5.70 Mbytes	47.8 Mbits/sec	0.345 ms	0/4129 (0%)	
2	1.00-2.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.161 ms	0/4315 (0%)	
3	2.00-3.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.408 ms	0/4317 (0%)	
4	3.00-4.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.154 ms	0/4316 (0%)	
5	4.00-5.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.301 ms	0/4314 (0%)	
6	5.00-6.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.038 ms	0/4319 (0%)	
7	6.00-7.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.270 ms	0/4314 (0%)	
8	7.00-8.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.055 ms	0/4317 (0%)	
9	8.00-9.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.385 ms	0/4316 (0%)	
10	9.00-10.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.178 ms	0/4318 (0%)	
11	10.00-11.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.130 ms	0/4316 (0%)	
12	11.00-12.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.377 ms	0/4316 (0%)	
13	12.00-13.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.235 ms	0/4315 (0%)	
14	13.00-14.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.075 ms	0/4317 (0%)	
15	14.00-15.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.118 ms	0/4312 (0%)	
16	15.00-16.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.257 ms	0/4321 (0%)	
17	16.00-17.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.234 ms	0/4317 (0%)	
18	17.00-18.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.130 ms	0/4314 (0%)	
19	18.00-19.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.239 ms	0/4319 (0%)	
20	19.00-20.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.342 ms	0/4315 (0%)	
21	20.00-21.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.393 ms	0/4317 (0%)	
22	21.00-22.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.263 ms	0/4315 (0%)	
23	22.00-23.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.163 ms	0/4317 (0%)	
24	23.00-24.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.354 ms	0/4316 (0%)	
25	24.00-25.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.141 ms	0/4316 (0%)	
26	25.00-26.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.063 ms	0/4318 (0%)	
27	26.00-27.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.355 ms	0/4314 (0%)	
28	27.00-28.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.401 ms	0/4317 (0%)	
29	28.00-29.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.380 ms	0/4316 (0%)	
30	29.00-30.00	sec 5.94 Mbytes	50.0 Mbits/sec	0.161 ms	0/4318 (0%)	
31	30.00-30.04	sec 263 Kbytes	49.9 Mbits/sec	0.427 ms	0/186 (0%)	

ID Interval Transfer Bitrate Jitter Lost/Total Delay

1 0.00-30.04 sec 179 Mbytes 49.5 Mbits/sec 0.427 ms 0/129457 (0%) z

receiver

Server listening on 5201

Accepted connection from 10.0.0.110, port 57168

ID	Interval	Transfer	Bitrate	Jitter	Lost/Total	Datagrams
1	0.00-1.00	sec 4.52 Mbytes	41.2 Mbits/sec	0.458 ms	209/3769 (5.5%)	
2	1.00-2.00	sec 4.71 Mbytes	39.5 Mbits/sec	0.341 ms	874/4256 (20%)	
3	2.00-3.00	sec 5.42 Mbytes	45.5 Mbits/sec	0.359 ms	474/4396 (11%)	
4	3.00-4.00	sec 5.57 Mbytes	46.7 Mbits/sec	0.575 ms	252/4286 (5.9%)	
5	4.00-5.00	sec 5.97 Mbytes	50.1 Mbits/sec	0.300 ms	140/4461 (3.1%)	
6	5.00-6.00	sec 5.38 Mbytes	45.1 Mbits/sec	0.638 ms	237/4332 (5.7%)	
7	6.00-7.00	sec 6.08 Mbytes	51.0 Mbits/sec	0.312 ms	240/4644 (5.2%)	
8	7.00-8.00	sec 4.67 Mbytes	39.2 Mbits/sec	0.324 ms	591/3973 (15%)	
9	8.00-9.00	sec 5.39 Mbytes	45.2 Mbits/sec	0.317 ms	536/4437 (12%)	
10	9.00-10.00	sec 5.55 Mbytes	46.5 Mbits/sec	0.363 ms	348/4366 (8%)	
11	10.00-11.00	sec 5.48 Mbytes	46.0 Mbits/sec	0.378 ms	307/4275 (7.2%)	
12	11.00-12.00	sec 5.37 Mbytes	45.0 Mbits/sec	0.494 ms	369/4236 (8.7%)	
13	12.00-13.00	sec 5.23 Mbytes	44.3 Mbits/sec	0.574 ms	541/4364 (12%)	
14	13.00-14.00	sec 5.74 Mbytes	48.2 Mbits/sec	0.515 ms	352/4512 (7.8%)	
15	14.00-15.00	sec 5.68 Mbytes	47.6 Mbits/sec	0.308 ms	88/4202 (2.1%)	
16	15.00-16.00	sec 5.72 Mbytes	48.0 Mbits/sec	0.280 ms	179/4315 (4%)	
17	16.00-17.00	sec 5.67 Mbytes	47.5 Mbits/sec	0.397 ms	174/4276 (4.1%)	
18	17.00-18.00	sec 5.62 Mbytes	47.1 Mbits/sec	0.282 ms	172/4240 (4.1%)	
19	18.00-19.00	sec 5.33 Mbytes	44.8 Mbits/sec	0.499 ms	428/4251 (10%)	
20	19.00-20.00	sec 5.30 Mbytes	44.5 Mbits/sec	0.314 ms	429/4467 (14%)	
21	20.00-21.00	sec 5.70 Mbytes	47.8 Mbits/sec	0.218 ms	368/4498 (8.2%)	
22	21.00-22.00	sec 5.96 Mbytes	50.0 Mbits/sec	0.342 ms	0/4317 (0%)	
23	22.00-23.00	sec 5.96 Mbytes	50.0 Mbits/sec	0.319 ms	0/4317 (0%)	
24	23.00-24.00	sec 5.96 Mbytes	50.0 Mbits/sec	0.228 ms	0/4315 (0%)	
25	24.00-25.00	sec 5.96 Mbytes	50.0 Mbits/sec	0.406 ms	0/4315 (0%)	
26	25.00-26.00	sec 5.96 Mbytes	50.0 Mbits/sec	0.230 ms	0/4319 (0%)	
27	26.00-27.00	sec 5.95 Mbytes	50.0 Mbits/sec	0.190 ms	0/4312 (0%)	
28	27.00-28.00	sec 5.97 Mbytes	50.0 Mbits/sec	0.326 ms	0/4320 (0%)	
29	28.00-29.00	sec 5.96 Mbytes	50.0 Mbits/sec	0.311 ms	0/4316 (0%)	
30	29.00-30.00	sec 5.96 Mbytes	50.0 Mbits/sec	0.156 ms	0/4314 (0%)	
31	30.00-30.05	sec 266 Kbytes	43.2 Mbits/sec	0.404 ms	0/188 (0%)	

ID Interval Transfer Bitrate Jitter Lost/Total Datagrams

1 0.00-30.05 sec 168 Mbytes 47.0 Mbits/sec 0.404 ms 7502/129485 (5.8%) receiver

Тестирование CoDel и ARIMA-pacing

```

Server listening on 5201
Accepted connection from 10.0.1.10, port 50710
[ S] local 10.0.2.10 port 5201 connected to 10.0.1.10 port 38658
[ ID] Interval      Transfer    Bitrate      Jitter    Lost/Total Datagrams
[ S] 0.00-1.00    sec 5.67 MBytes 47.5 Mbit/sec 0.358 ms  0/4105 (0%)
[ S] 1.00-2.00    sec 5.97 MBytes 50.0 Mbit/sec 0.163 ms  0/4322 (0%)
[ S] 2.00-3.00    sec 5.98 MBytes 50.0 Mbit/sec 0.251 ms  0/4311 (0%)
[ S] 3.00-4.00    sec 5.96 MBytes 50.0 Mbit/sec 0.299 ms  0/4316 (0%)
[ S] 4.00-5.00    sec 5.97 MBytes 50.0 Mbit/sec 0.307 ms  0/4320 (0%)
[ S] 5.00-6.00    sec 5.96 MBytes 50.0 Mbit/sec 0.436 ms  0/4316 (0%)
[ S] 6.00-7.00    sec 5.96 MBytes 50.0 Mbit/sec 0.204 ms  0/4318 (0%)
[ S] 7.00-8.00    sec 5.96 MBytes 50.0 Mbit/sec 0.262 ms  0/4316 (0%)
[ S] 8.00-9.00    sec 5.96 MBytes 50.0 Mbit/sec 0.414 ms  0/4316 (0%)
[ S] 9.00-10.00   sec 5.96 MBytes 50.0 Mbit/sec 0.368 ms  0/4317 (0%)
[ S] 10.00-11.00  sec 5.96 MBytes 50.0 Mbit/sec 0.401 ms  0/4314 (0%)
[ S] 11.00-12.00  sec 5.96 MBytes 50.0 Mbit/sec 0.173 ms  0/4318 (0%)
[ S] 12.00-13.00  sec 5.96 MBytes 50.0 Mbit/sec 0.272 ms  0/4315 (0%)
[ S] 13.00-14.00  sec 5.96 MBytes 50.0 Mbit/sec 0.385 ms  0/4313 (0%)
[ S] 14.00-15.00  sec 5.96 MBytes 50.0 Mbit/sec 0.187 ms  0/4316 (0%)
[ S] 15.00-16.00  sec 5.96 MBytes 49.8 Mbit/sec 0.277 ms  0/4316 (0%)
[ S] 16.00-17.00  sec 5.97 MBytes 50.2 Mbit/sec 0.151 ms  0/4321 (0%)
[ S] 17.00-18.00  sec 5.96 MBytes 50.0 Mbit/sec 0.282 ms  0/4316 (0%)
[ S] 18.00-19.00  sec 5.96 MBytes 50.0 Mbit/sec 0.345 ms  0/4313 (0%)
[ S] 19.00-20.00  sec 5.96 MBytes 50.0 Mbit/sec 0.320 ms  0/4315 (0%)
[ S] 20.00-21.00  sec 5.96 MBytes 50.0 Mbit/sec 0.316 ms  0/4313 (0%)
[ S] 21.00-22.00  sec 5.96 MBytes 50.0 Mbit/sec 0.421 ms  0/4315 (0%)
[ S] 22.00-23.00  sec 5.96 MBytes 50.0 Mbit/sec 0.334 ms  0/4312 (0%)
[ S] 23.00-24.00  sec 5.96 MBytes 50.0 Mbit/sec 0.408 ms  0/4315 (0%)
[ S] 24.00-25.00  sec 5.96 MBytes 50.0 Mbit/sec 0.341 ms  0/4315 (0%)
[ S] 25.00-26.00  sec 5.96 MBytes 50.0 Mbit/sec 0.342 ms  0/4313 (0%)
[ S] 26.00-27.00  sec 5.96 MBytes 50.0 Mbit/sec 0.267 ms  0/4315 (0%)
[ S] 27.00-28.00  sec 5.96 MBytes 50.0 Mbit/sec 0.458 ms  0/4313 (0%)
[ S] 28.00-29.00  sec 5.97 MBytes 50.0 Mbit/sec 0.174 ms  0/4320 (0%)
[ S] 29.00-30.00  sec 5.96 MBytes 49.9 Mbit/sec 0.173 ms  0/4309 (0%)
[ S] 30.00-30.05  sec 5.97 MBytes 49.1 Mbit/sec 0.355 ms  0/210 (0%)
[ ID] Interval      Transfer    Bitrate      Jitter    Lost/Total Datagrams
[ S] 0.00-30.05   sec 179 MBytes 49.9 Mbit/sec 0.355 ms  0/125852 (0%) receiver
    
```

```

Server listening on 5201
Accepted connection from 10.0.1.10, port 36308
[ S] local 10.0.2.10 port 5201 connected to 10.0.1.10 port 36311
[ ID] Interval      Transfer    Bitrate      Jitter    Lost/Total Datagrams
[ S] 0.00-1.00    sec 5.65 MBytes 47.4 Mbit/sec 0.394 ms  0/4092 (0%)
[ S] 1.00-2.00    sec 5.96 MBytes 50.0 Mbit/sec 0.379 ms  0/4319 (0%)
[ S] 2.00-3.00    sec 5.95 MBytes 49.8 Mbit/sec 0.316 ms  0/4311 (0%)
[ S] 3.00-4.00    sec 5.97 MBytes 50.0 Mbit/sec 0.307 ms  0/4321 (0%)
[ S] 4.00-5.00    sec 5.96 MBytes 50.0 Mbit/sec 0.324 ms  0/4317 (0%)
[ S] 5.00-6.00    sec 5.96 MBytes 50.0 Mbit/sec 0.210 ms  0/4318 (0%)
[ S] 6.00-7.00    sec 5.96 MBytes 50.0 Mbit/sec 0.369 ms  0/4316 (0%)
[ S] 7.00-8.00    sec 5.96 MBytes 50.0 Mbit/sec 0.366 ms  0/4314 (0%)
[ S] 8.00-9.00    sec 5.97 MBytes 50.1 Mbit/sec 0.286 ms  0/4320 (0%)
[ S] 9.00-10.00   sec 5.96 MBytes 50.0 Mbit/sec 0.411 ms  0/4314 (0%)
[ S] 10.00-11.00  sec 5.96 MBytes 50.0 Mbit/sec 0.415 ms  0/4314 (0%)
[ S] 11.00-12.00  sec 5.96 MBytes 50.0 Mbit/sec 0.338 ms  0/4316 (0%)
[ S] 12.00-13.00  sec 5.96 MBytes 50.0 Mbit/sec 0.356 ms  0/4317 (0%)
[ S] 13.00-14.00  sec 5.96 MBytes 50.0 Mbit/sec 0.319 ms  0/4315 (0%)
[ S] 14.00-15.00  sec 5.96 MBytes 50.0 Mbit/sec 0.350 ms  0/4319 (0%)
[ S] 15.00-16.00  sec 5.96 MBytes 50.0 Mbit/sec 0.273 ms  0/4317 (0%)
[ S] 16.00-17.00  sec 5.96 MBytes 50.0 Mbit/sec 0.397 ms  0/4317 (0%)
[ S] 17.00-18.00  sec 5.96 MBytes 50.0 Mbit/sec 0.334 ms  0/4314 (0%)
[ S] 18.00-19.00  sec 5.97 MBytes 50.0 Mbit/sec 0.406 ms  0/4320 (0%)
[ S] 19.00-20.00  sec 5.96 MBytes 50.0 Mbit/sec 0.287 ms  0/4314 (0%)
[ S] 20.00-21.00  sec 5.96 MBytes 49.9 Mbit/sec 0.324 ms  0/4313 (0%)
[ S] 21.00-22.00  sec 5.96 MBytes 50.1 Mbit/sec 0.289 ms  0/4317 (0%)
[ S] 22.00-23.00  sec 5.96 MBytes 50.0 Mbit/sec 0.278 ms  0/4316 (0%)
[ S] 23.00-24.00  sec 5.96 MBytes 50.0 Mbit/sec 0.324 ms  0/4317 (0%)
[ S] 24.00-25.00  sec 5.96 MBytes 50.0 Mbit/sec 0.286 ms  0/4316 (0%)
[ S] 25.00-26.00  sec 5.96 MBytes 50.0 Mbit/sec 0.275 ms  0/4318 (0%)
[ S] 26.00-27.00  sec 5.96 MBytes 50.0 Mbit/sec 0.385 ms  0/4316 (0%)
[ S] 27.00-28.00  sec 5.96 MBytes 50.0 Mbit/sec 0.296 ms  0/4315 (0%)
[ S] 28.00-29.00  sec 5.96 MBytes 50.0 Mbit/sec 0.200 ms  0/4315 (0%)
[ S] 29.00-30.00  sec 5.96 MBytes 50.0 Mbit/sec 0.446 ms  0/4318 (0%)
[ S] 30.00-30.06  sec 5.98 MBytes 49.7 Mbit/sec 0.318 ms  0/216 (0%)
[ ID] Interval      Transfer    Bitrate      Jitter    Lost/Total Datagrams
[ S] 0.00-30.06   sec 179 MBytes 49.9 Mbit/sec 0.318 ms  0/125852 (0%) receiver
    
```

Сравнение алгоритмов

Алгоритм	Джиттер(мс)	Потери пакетов(%)
FIFO	0.427	0
RED	0.404	5.8
CoDel	0.355	0
ARIMA-pacing	0.318	0

Результаты работы

- В ходе работы создан сетевой маршрутизатор с динамически загружаемыми планировщиками очереди. Реализована многопоточная архитектура с отдельными потоками приёма и отправки для дуплексных направлений, плагиновая система загрузки алгоритмов, захват и отправка сырых IP-пакетов через RAW-сокеты, фильтрация трафика на уровне приложения. Тестирование в среде GNS3 с генерацией трафика утилитой iperf3 подтвердило работоспособность продукта.
- В ходе работы реализованы и исследованы четыре алгоритма управления очередью: FIFO, RED, CoDel и ARIMA-pacing. Экспериментальные результаты показали, что ARIMA-pacing обеспечивает наилучший показатель джиттера — 0,318 мс, что на 25,5 % ниже FIFO и на 10,4 % ниже CoDel, при сохранении пропускной способности 49,9 Мбит/с и отсутствии потерь. Это подтверждает эффективность применения прогнозирования временных рядов в задачах управления очередью.

Спасибо за внимание

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.31.001 ПЗ

Лист

142