Ордена Трудового Красного Знамени
федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

На правах рукописи

Краснова Ирина Артуровна

# Динамическая классификация потоков трафика на основе машинного обучения для обеспечения качества обслуживания в мультисервисной программно-конфигурируемой сети

Специальность 05.12.13 – Системы, сети и устройства телекоммуникаций

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
кандидат технических наук, доцент
Деарт Владимир Юрьевич

Москва, 2021

# Оглавление

# Введение

**Актуальность темы исследования.** В последнее время все большее распространение получают программно-конфигурируемые *SDN*-сети (*Software-Defined Networking*), архитектура которых подразумевает отделение плоскости управления от плоскости передачи данных и формирование блока единого централизованного управления сетью. Такая концепция позволяет получить определенные преимущества в организации и управлении информационными потоками, а также добиться большей гибкости в доступе к ресурсам сетей и устройств, и более динамичного развития существующей сетевой инфраструктуры, в отличие от традиционного подхода к построению сетей.

Мультисервисные *SDN*-сети предлагают абоненту целый спектр разнообразных услуг, среди которых выделяют голосовую связь, видеосвязь, видеоконференцию, передачу данных и т.д., а кроме того, позволяют легко добавлять новые приложения и изменять существующие. Каждое из созданных приложений требует обеспечения определенного уровня качества обслуживания *QoS* (*Quality of Service*), что с увеличением числа потребителей и услуг становится все сложнее.

Традиционные механизмы *Differential Services* (услуги с дифференцированным обслуживанием) позволяют обеспечивать определенный уровень *QoS* в рамках одного из классов обслуживания, к которым могут относить тип приложения (*Skype, Telnet* и т.д.), тип передаваемой информации (голос, видео, данные), характер передачи трафика (*Elephant* и *Mice*–потоки - очень большие и очень маленькие потоки соответственно, интерактивный/не интерактивный трафик) и т.д. Для известных оператору сети источников трафика возможна автоматическая или предварительная статическая разметка пакетов с указанием принадлежности сервиса, который используется для дифференцированного управления трафиком. В случаях поступления в сеть пакетов, неучтенных с точки зрения архитектуры сети либо созданных неизвестным для оператора источником (приложением или сервисом), они

обрабатываются по принципу негарантированной доставки (*Best Effort*), который не обеспечивает *QoS* на необходимом уровне. Проблема имеет особенное значение для *SDN*-сетей, в которых регулярно появляются новые приложения, а топология сети динамически меняется.

**Степень разработанности темы.**

Предыдущие подходы к классификации потоков в традиционных сетях, основывались на общеизвестном списке *TCP* и *UDP*-портов, но с появлением динамически изменяющихся портов применение такого метода стало невозможным.

Широко известная технология *DPI* (*Deep Packet Inspection*) позволяет проводить «глубокий» анализ заголовков пакетов на верхних уровнях модели ЭМВОС (*OSI*). Но с помощью системы *DPI* тоже не всегда удается выявить характер потока данных, например, в случаях зашифрованного или туннелированного трафика.

В последнее время в телекоммуникациях все чаще эффективно применяются методы интеллектуального анализа данных, в особенности методы машинного обучения (*Machine Learning, ML*), для решения широкого круга задач, в т.ч. и для классификации трафика.

Исследования и анализ основных работ по классификации методами *ML*, представленные *Гетьманом А.И., Маркиным Ю.В., Ванюшиной А.В., Xie J., Huang N., Zhao D., Perera P., Zhang X., Dong Y., Zhang C., Wang W., Latah M., Pekar A., Habibi L.A., Bakker J.N.* позволили выделить **основные проблемы**, существующие на данный момент:

— проблему с доступом к параметрам пакетов, применяемым в матрице признаков — для большинства работ требуется наличие полной информации о потоке или подробной информации уровня приложений, что означает **ограниченность применения классификатора** для защищенных потоков и **невозможность работы в режиме реального времени**;

— тяжеловесные алгоритмы мониторинга сети, требующие постоянных действий «запрос» - «ответ» от коммутаторов и контроллеров и вызывающие **большую нагрузку** на сеть и сетевые элементы;

— **отсутствие** функциональной **возможности добавления новых классов** в существующую модель классификатора делает невозможным его работу в динамически изменяющихся сетях, таких как SDN.

Российские ученые: *Бурлаков М.Е., Осипов М.Н., Шелухин О.И., Ерохин С.Д.* и зарубежные исследователи: *Alothman B., Gombault S., Toker M., Knapskog S.J., Buczak L., Hodo E., Knapskog S.J., Hamilton A.W., Tachtatzis C., Atkinson R.C.* и др. частично рассматривают эти вопросы, но их **работы сосредоточены** в основном **для** обеспечения **сетевой безопасности**, в то время как классификация трафика с целью определения *QoS* имеет значительные отличия, к которым следует отнести маркировку классов с целью присвоения *QoS*, различия при формировании матрицы признаков и при предварительной обработке данных. Таким образом, разработка методов **классификации** трафика **с целью обеспечения *QoS* требует** проведения исследований и **разработки** других **алгоритмов.**

*Примечание:* здесь и далее под «классификацией трафика» подразумевается «классификация трафика с целью обеспечения *QoS*», если не сказано иное.

**Объектом исследования** являются потоки трафика в мультисервисной *SDN*-сети.

**Предметом исследования** являются характеристики потоков трафика в мультисервисной *SDN*-сети.

**Целью диссертационного исследования** является разработка метода динамической классификации потоков трафика на основе машинного обучения для обеспечения качества обслуживания в мультисервисной *SDN*-сети в режиме реального времени.

**Задачи диссертационного исследования,** решаемые для достижения поставленной цели:

1. Формирование **матрицы признаков** для классификации потоков трафика в режиме реального времени.

2. Разработка **статической модели классификации трафика** методами машинного «обучения с учителем» на основе созданной матрицы признаков.

3. Исследование и разработка **модели кластеризации трафика** методами машинного «обучения без учителя».

4. Создание алгоритма **сбора статистических характеристик** потоков трафика в *SDN*-сетях для сформированной матрицы признаков.

5. Создание эффективного метода **динамической классификации** трафика на основе разработанных моделей, обладающего способностью обнаруживать новые классы и работающего в режиме реального времени.

**Научная новизна.** Научная новизна работы заключается в создании новой модели классификации потоков трафика, отличающейся от ранее представленных следующими ключевыми особенностями:

1. Разработана принципиально **новая матрица признаков**, в которой индивидуальные статистические характеристики каждого из первых 15 пакетов потока предлагается использовать как отдельные признаки, что позволяет применять ее для активных потоков в режиме реального времени, в то время как **общепринятые подходы** предполагают расчет характеристик на основе данных всего потока.

2. Разработана **статическая модель классификации трафика**, **отличающаяся от известных тем, что** включает в себя блок настройки гиперпараметров ансамблевых алгоритмов на основе «решающего дерева» (глубина дерева, количество деревьев, минимальное количество классов в узле для разветвления и т.д.) и блок предварительной обработки данных, основанный на комплексном подходе с использованием квантильной трансформации, параметрического преобразования Йео-Джонсона и работы с выбросами; что позволяет повысить точность классификации и расширить область применения алгоритма *XGBoost* для задач классификации трафика.

3. Доказано **повышение эффективности кластеризации трафика** за счет использования предварительно рассчитанных матриц расстояний на основе методов *Extremely Randomized Trees* и *Random Forest*. В работе **впервые применяется данный подход** для матрицы признаков режима реального времени, в предыдущих исследованиях кластеризация рассматривается только для применения в моделях вне режима реального времени.

4. Сформирована **система организации памяти** *P4*-коммутатора, организованная за счет выделения ячеек памяти - регистров и назначения им информационного и управленческого функционала. Эта система **отличается от других** своей гибкостью — она позволяет хранить и передавать на контроллер статистическую и идентификационную информацию о любом пакете и только по мере необходимости (например, после накопления первых 10 пакетов), что снижает нагрузку на сеть и устройства по сравнению с традиционными вариантами мониторинга сетевых элементов.

5. Создана **динамическая модель классификации трафика**, полученная за счет объединения точности и скорости работы методов «обучения с учителем» с возможностью образования новых кластеров за счет методов «обучения без учителя», что позволяет ей, **в отличие от других,** обладать одновременно **тремя свойствами**: проводить классификацию для целей обеспечения *QoS*, работать в режиме реального времени и добавлять новые классы в существующую систему.

**Теоретическая и практическая значимость работы.**

Теоретическая значимость исследования обоснована тем, что проведена модернизация существующих математических моделей, позволяющая эффективно применять методы машинного обучения для классификации потоков трафика для обеспечения *QoS* в режиме реального времени; применительно к проблематике диссертации результативно использован комплекс существующих методов машинного обучения; впервые раскрыта и доказана эффективность применения метода *XGBoost* для классификации трафика; изложен метод

построения модели классификатора трафика с возможностью добавления новых классов.

Практическую значимость исследования представляют следующие его элементы: статическая модель классификации трафика может применяться на участках сетей с относительно постоянным составом приложений, в т.ч. на традиционных сетях; динамическая модель классификации трафика может применяться для динамически изменяющихся сетей, в т.ч. *SDN*-сетей; метод хранения, сбора и обработки статистической информации может применяться в программируемых *P4*-коммутаторах не только для классификации, но и для других целей мониторинга.

Результаты научно-квалифицированной работы используются при разработке и реализации проектов в **ЗАО «ИнформИнвестГрупп»** и **ООО НПФ «Гранч»**, а также внедрены в учебный процесс кафедры **СиСФС МТУСИ**, что подтверждается соответствующими актами.

**Личный вклад.** Все основные научные положения, промежуточные выводы, представленные в диссертации, получены автором лично.

**Методология и методы исследования.** Для решения поставленных задач применялись методы машинного обучения, математической статистики, теории вероятностей, линейной алгебры, натурного эксперимента и имитационного моделирования.

**Работа соответствует паспорту специальности** 05.12.13 «Системы, сети и устройства телекоммуникаций» по части вопросов комплексного решения технических проблем, задач и вопросов систем и устройств телекоммуникаций. Основные результаты диссертации были получены при работе в следующих областях:

— при исследовании процессов представления информации и создания новых соответствующих алгоритмов и процедур (п. 2) был разработан алгоритм представления данных об информационных потоках трафика в виде матрицы признаков для классификации;

— при разработке эффективных путей развития и совершенствования архитектуры сетей и систем телекоммуникаций и входящих в них устройств (п. 3) была разработана модель классификации трафика с целью поддержания *QoS* в режиме реального времени;

— при разработке новых методов дифференцированного доступа абонентов к ресурсам сетей, систем и устройств телекоммуникаций (п. 5) был предложен новый алгоритм сбора индивидуальной статистической информации о пакетах для *P4*-коммутаторов.

**Положения, выносимые на защиту.**

1. **Матрица признаков** для классификации трафика с целью поддержания *QoS*, в которой признаками являются индивидуальные статистические параметры первых 10-15 пакетов, такие как длина и межинтервальное время прихода пакета на интерфейс, **повышает точность** классификации **на 10-25%** для *TCP* –потоков и до **2%** для *UDP*-потоков по сравнению с другими известными подходами. Структура такой матрицы позволяет эффективно применять классификацию к активным потокам в режиме реального времени и является **инвариантной** по отношению к разным типам потоков трафика.

2. **Статическая модель классификации** трафика на основе созданной матрицы признаков **повышает точность** классификации на **15-25%** для метода **«случайного леса»** и на **30-40%** для **«градиентного бустинга»** по сравнению с другими распространенными подходами за счет использования квантильной трансформации, параметрического преобразования Йео-Джонсона, удаления выбросов и настройки гиперпараметров.

3. **Модель кластеризации трафика**, адаптированная к созданной матрице признаков, **достигает значений согласованного индекса Рэнда 90-100%** за счет применения в процессе кластеризации матрицы расстояний, предварительно рассчитанной на основе результатов классификации потоков методами *Extremely Randomized Trees*. Показано также, что наиболее

распространенные и **стандартные подходы** к расчету матриц расстояний, такие как расстояния *Евклида* и *Манхэттена*, **оказались непригодными** к применению в таких условиях.

4. **Алгоритм гибкого сбора статистической информации о пакетах** в *P4*-коммутаторах, основанный на разработанной системе организации памяти, **позволяет хранить и передавать** на контроллер статистическую и идентификационную **информацию о любом пакете** и только **по мере необходимости** (например, после накопления первых 10 пакетов), что **снижает долю** передаваемой **служебной информации**, создающую дополнительную нагрузку на сеть и устройства, в **4-4,5** раза по сравнению с традиционными вариантами полного мониторинга сетевых элементов.

5. Расширенная **динамическая модель классификации** трафика **для целей** обеспечения ***QoS,*** на основе комбинирования методов классификации и кластеризации, **работает в режиме реального времени**, используя скорость и точность работы методов «обучения с учителем», и **добавляет новые классы** за счет методов «обучения без учителя».

**Степень достоверности и апробация результатов.**

Достоверность результатов обеспечивается за счет корректного применения математического аппарата, программного обеспечения и подтверждается результатами расчетов и моделирования.

Основные результаты работы обсуждались на научном межвузовском семинаре *«Современные телекоммуникации и математическая теория телетрафика (СТ и МТТ) № 59»* (Москва, 2021) и **восьми** международных научных конференциях: *«28$^{th}$ Conference of Open Innovations Association (FRUCT)»*, (Москва, 2021), *«The International Science and Technology Conference „Modern Network Technologies, MoNeTec - 2020“»* (Москва, 2020), *«The First International Symposium on Computer Science, Digital Economy and Intelligent Systems (CSDEIS2019)»* (Москва, 2019), *«The Second International Symposium on Computer Science, Digital Economy and Intelligent Systems (CSDEIS2020)»* (Москва, 2020), *«The Fourth International Conference of Artificial*

*Intelligence, Medical Engineering, Education (AIMEE2020)*» (Москва, 2020), *«XI Международной отраслевой научно-технической конференции „Технологии информационного общества“»* (Москва, 2017), *«Международной научно-технической конференции „Информационные технологии и математическое моделирование систем 2019“ (ИТММС 2019)»* (Одинцово, 2019) и «X *Московской научно-практической конференции „Студенческая наука -2015“*» (Москва, 2015).

По теме диссертации опубликовано **12** печатных работ, из них **5** в научных изданиях, индексированных в международных наукометрических базах, в т.ч. ***WoS***, ***Scopus*** и ***Springer***, **3** в ведущих рецензируемых научных журналах, рекомендованных ***ВАК***, и **1** учебное пособие.

**Объем и структура работы.** Диссертация изложена на 194 страницах, включает в себя 81 рисунок, 33 таблицы и состоит из введения, пяти разделов, заключения, списка источников из 164 наименований, списка сокращений, и 4-х приложений.

# Раздел 1. Анализ перспективных подходов и исследований по классификации потоков трафика для поддержания качества обслуживания методами машинного обучения в SDN-сетях

## 1.1. Особенности классификации трафика в SDN-сетях

Программно-конфигурируемые *SDN*-сети (*Software-Defined Networking*, [1]) – способ организации сетей, при котором плоскость управления (*Control Plane*) отделена от плоскости передачи данных (*Data Plane*) и представляет собой единую структуру (*SDN*-контроллер), способную предоставить централизованный контроль и доступ ко всем сетевым элементам. Посредством плоскости приложений (*Application Plane*), администратор сети способен повлиять на обработку трафика, создавая новые приложения и взаимодействуя через *API*-интерфейс (*Application Programming Interface*) с контроллером (Рисунок 1.1) [2-3].

Становление и развитие архитектуры *SDN*-сетей стимулирует появление все новых подходов к классификации потоков трафика в сетях на основе методов машинного обучения (*Machine Learning, ML).*

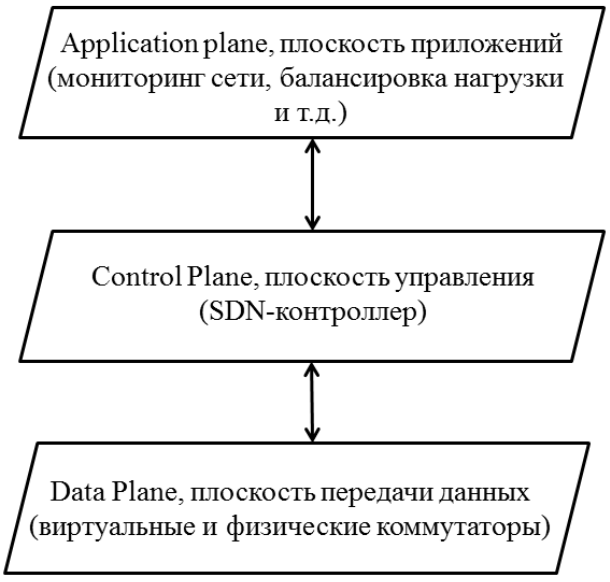



Рисунок 1.1 - Архитектура *SDN*-сетей

Преимущества *SDN*-сетей, используемые при классификации трафика [4]:

— централизованный мониторинг сети с любых узлов;

— расширенная информация о потоках и пакетах по портам, узлам, сетям, в т.ч. информация о каждом пакете, размере, времени отправки, флагах *TCP*, заголовках приложений, времени прохождения пакетом определенного этапа обработки коммутатором (*P4*-сети, *Programming Protocol-Independent Packet Processors*) и т.д.;

— централизованный анализ данных без дополнительной нагрузки на сетевые элементы;

— управление сетью на основе множества факторов;

— исследование трафика в режиме реального времени;

— возможность создания собственных протоколов и приложений для мониторинга, классификации и принятия решений по управлению трафиком.

В *SDN*-сетях имеется возможность проводить интеллектуальную классификацию трафика на разных плоскостях. В [5] классификация трафика проводится в плоскости *Data Plane* и позволяет выявить *Elephant*-потоки (очень большие потоки, занимающие большую долю пропускной полосы). В [6] потоки классифицируются в плоскости *Control Plane*, а в [7-8] в плоскости *Application Plane*. В [9] создали свою дополнительную плоскость, в которой проводится интеллектуальная обработка данных о потоках.

Основные сферы применения методов *ML* в *SDN*-сетях:

1) классификация потоков трафика [10-27];

2) динамическое определение оптимального маршрута с учетом состояния соединений и сетевых элементов [28-29];

3) прогнозирование параметров *QoS* / *QoE* (*Quality of Experience,* качество восприятия) на основе имеющейся статистической информации [30-33];

4) управление ресурсами [29; 34-36];

5) управление политиками безопасности, в т.ч. обнаружение вторжений [37-38];

6) проведение статистических исследований, предсказание занятости каких-либо каналов связи или использование определенных приложений [24;29].

Существует классификация трафика в режиме реального времени и классификация трафика вне режима реального времени.

Классификация трафика в режиме реального времени чаще всего применяется с целью маркировки трафика и поддержки параметров *QoS* либо для систем обнаружения вторжений. Особенности такой классификации заключаются в следующем:

1. Системы, работающие в таком режиме, должны быть достаточно быстродействующими, т.к. результат классификации должен быть дан достаточно быстро, иначе ответ будет уже неактуален.

2. Классификация проводится только по $n$ первым пакетам или $t$ первым мс, где параметры $n$ и $t$ стремятся минимизировать. Невозможность просмотра всего потока накладывает определённые ограничения на матрицу признаков для классификации, в частности, в качестве признаков не могут быть выбраны длительность всего потока, количество пакетов в потоке, средние параметры по потоку.

3. Методика сбора информации о потоках должна быть достаточно гибкой и способной снимать и оперативно передавать лишь необходимое количество информации, не создавая дополнительных нагрузок на сеть.

4. Результаты классификации должны быть достаточно точными, что делает невозможным применение непроверенных и слабых подходов и методов.

5. Классификатор должен также определять неизвестные ему приложения.

Классификация трафика вне режима реального времени может применяться, например, для статистических исследований, с целью выявления наиболее популярных приложений и услуг. Такие методы позволяют накапливать всю информацию о потоке, но делают невозможным их применение в режиме реального времени.

## 1.2. Этапы классификации потоков трафика в режиме реального времени

Под классификацией потоков трафика подразумевается разделение различных образцов трафика (потоков, сессий, групп пакетов) на определенные классы (группы, категории).

Вне зависимости от того, какой механизм был разработан и какие методы машинного обучения в нем принимались, при его верификации, база данных для классификации состоит из двух частей: набора образцов трафика и набора соответствующих им маркеров, определяющих принадлежность образца к какому-либо классу. В самом разработанном алгоритме при этом может и не быть маркеров принадлежности класса (методы «обучения без учителя», кластеризация и т.д.).

На Рисунке 1.2 показаны этапы верификации метода классификации потоков трафика в режиме реального времени в сетях *SDN* и основные вопросы, возникающие на различных этапах реализации метода:

1. Формирование базы данных для классификации: создание базы данных, разметка базы данных и сбор статистической информации о потоках трафика.

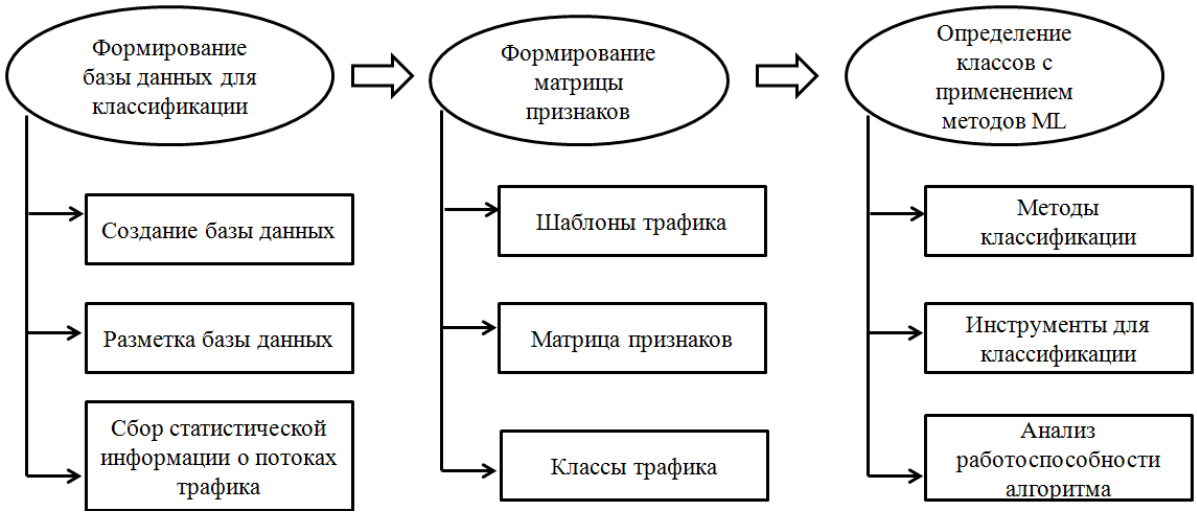



Рисунок 1.2 - Этапы классификации трафика в режиме реального времени

2. Формирование матрицы признаков: определение шаблонов трафика, построение матрицы признаков и определение классов трафика как результата работы алгоритма.

3. Определение классов с применением методов *ML*: методы классификации, инструменты для интеллектуального анализа данных и анализ работоспособности предложенного алгоритма.

### 1.2.1. Методы формирования базы данных для классификации

Одной из сложнейших задач при классификации сетевого трафика, замедляющих развитие этого направления в целом, является формирование базы данных трафика, т.к. сетевой трафика разнородный, быстроменяющийся, сложный и закрытый в целях безопасности от сторонних наблюдателей. Не существует уникальных общедоступных баз данных. Все попытки создания подобной базы сопряжены с определенными трудностями, из-за которых в результате базы получаются с существенными недостатками, ограничивающими область их применения. В итоге большинство исследователей проводят эксперименты на различных базах данных собранных в разное время при различных условиях и итоги их работы довольно сложно сопоставить и сравнить между собой.

#### ***1.2.1.1. Способы создания базы данных для классификации (Рисунок 1.3):***

I. Моделирование потоков трафика в условиях лабораторного стенда (инвазивный трафик):

1. Имитация работы приложений с использованием различных генераторов трафика. Существует множество приложений и устройств, которые способны смоделировать трафик (например, *D-ITG* - генератор [39]). В данном случае, заранее известен трафик, который проходит классификацию, его легко промаркировать и выделить среди остальных потоков, что является неоспоримым

преимуществом данного способа. Недостатком же является излишняя «искусственность» сетей, при которой зачастую ситуация со смоделированным трафиком очень сильно отличается от реальной работы сети. Кроме того, уровень приложений такого трафика обычно заполняется случайными либо повторяющимися значениями, что исключает возможность применения классификации трафика на основе данных прикладного уровня.

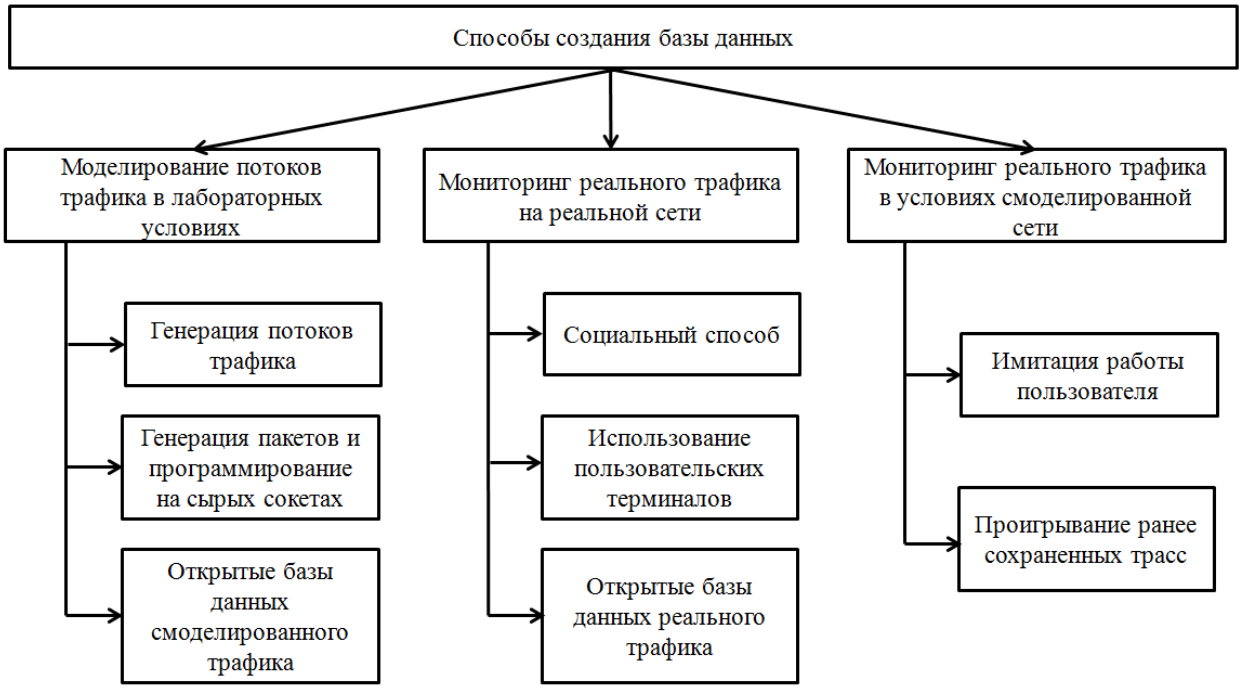



Рисунок 1.3 - Способы создания базы данных

2. Программирование на сырых сокетах (*raw socket*) и создание отдельных пакетов на основе *Python Scapy* [40]. Сырые сокеты – специальный *API* - интерфейс, который позволяет принимать и передавать необработанные пакеты без добавления в них каких-либо дополнительных заголовков. Преимуществом метода является неограниченная возможность по созданию трафика, т.к. могут быть получены абсолютно любые пакеты и трассы. Недостатками так же, как и в использовании генераторов трафика, выступает нереалистичность полученных баз трафика, а также повышенные технические сложности по организации эксперимента.

3. Использование открытых баз данных. В открытом доступе находится довольно большое количество открытых баз данных [41-43]. Они могут быть представлены как в формате трасс трафика (*.pcap*), так и в формате матриц признаков, рассчитанных на основе каких-то определенных потоков. Преимуществом способа является простота использования и возможность быстрого доступа к разнообразным сервисам, которые зачастую очень сложно даже смоделировать на тестовом стенде. Основными недостатками является отсутствие маркировки потоков, ограниченность представленных сервисов, фиксированные лабораторные условия и быстрое «устаревание» информации.

II. Мониторинг реального трафика на реальной сети (неинвазивный трафик):

1. Социальный способ, с привлечением партнеров, учащихся и сотрудников университетов или пользователей созданной сети; в коммерческой среде может мотивироваться получением определенных бонусов/баллов либо денежных вознаграждений клиентам. Преимуществами такой сети является, безусловно, реалистичность и актуальность собранной базы данных, большие объемы трафика, возможность получения разнообразного вида трафика, который не всегда может быть представлен или учтен в условиях лабораторных стендов, а также такой подход не вносит дополнительных нагрузок на сеть. Недостатки такого метода: отсутствие маркировки как таковой; нестабильность условий - невозможность проведения экспериментов с равными условиями; сложность организации социальной структуры - привлечение большого количества людей и как следствие - затруднения при повторной организации подобных экспериментов; политики конфиденциальности – передача трасс трафика проходящего по сети даже с целью исследования является небезопасным, поэтому над трассами проводятся процедуры анонимизации – удаления полезной информации и шифрование трассы, что ограничивает варианты применения методов машинного обучения.

2. Использование собственных пользовательских терминалов. В некоторых исследованиях можно встретить базы данных, полученные при

помощи мониторинга собственных пользовательских терминалов, в качестве которых обычно выступают ПК и ноутбуки одного или нескольких участников эксперимента. Такой подход намного проще организовать в социальном и техническом плане, в некоторых вариантах возможна маркировка трафика, но по сравнению с предыдущим способом получается значительно меньшая по объему и худшая по качеству база данных, а также подобные эксперименты позволяют снимать трафик только в одной точке сети – на оконечном устройстве, не имея представления о том, что происходит в остальных узлах сети.

3. Использование выложенных в интернете баз данных трафика. Подобные базы обладают теми же недостатками, что и аналогичные базы данных с моделированными потоками [44].

III. Мониторинг реального трафика в условиях смоделированной сети:

1. Получение потоков трафика с использованием различных приложений и программ, имитирующих работу пользователей. Ввиду сложности создания автоматизированных систем с использованием голосового трафика, такие исследования включают в себя в основном *Web*-трафик, *DNS* и *FTP*.

2. Проигрывание ранее записанных трасс трафика, полученных в реальных сетях. Например, *tcpreplay* [45] позволяет проигрывать трассы по их *.pcap* – записям. Таким образом, зная сценарии определенных реальных трасс, можно воссоздать новую трассу в других условиях. Однако, остается некоторая неопределенность при использовании такого подхода: с одной стороны, имеется определенный сценарий поведения, которым руководствуется один из узлов, а с другой стороны – трасса получена с одними условиями, а воспроизводится с другими, что влияет на ее параметры, например, *NAT*, *VLAN*, *TCP/UDP*-порты и т.д.

#### *1.2.1.2. Разметка базы данных*

Помимо непосредственного сбора базы данных трафика необходимо проводить и ее разметку, т.е. определять принадлежность образца к определенному классу. Способы разметки базы данных (Рисунок 1.4):

I. Априорные способы – способы разметки трафика непосредственно во время сбора трафика или до его прохождения по сети:

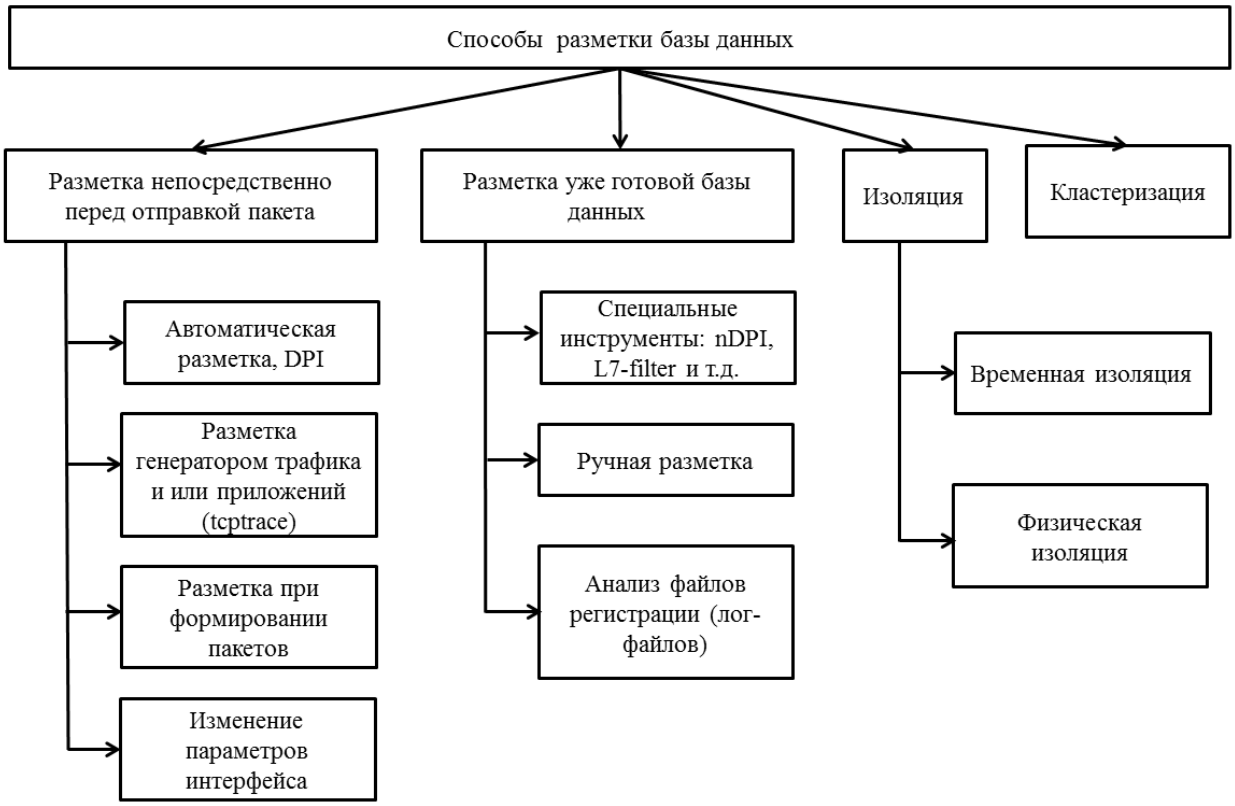



Рисунок 1.4 - Способы разметки базы данных

1. Автоматическая разметка с использованием специальных инструментов, включающих в себя разметку по портам, разметку с помощью *DPI* [46], системы определения сервиса на основе ответов *DNS* [47] и т.д. Такие методы можно применять на реальных и достаточно больших сетях, но они имеют ряд недостатков: маркировка по портам достаточно простая, но в сетях с динамическим изменением портов не очень эффективная. Системы с *DPI* технически более сложно реализуемы и не работают во многих случаях, например, с зашифрованным трафиком.

2. Разметка перед отправкой пакетов в моделируемых трассах – ряд генераторов трафика позволяет менять поля *TOS* (*Type Of Service*) / *DSCP* (*Differentiated Services Code Point*) у отправляемых пакетов. Также подобный способ может применяться при проигрывании известных трасс с изменением

соответствующих полей. Другой разновидностью этого способа является разметка потоков с помощью известных запущенных процессов или с помощью специальных приложений, таких как *tcptrace* [48].

3. Разметка при формировании пакетов используется в случаях с программированием на сырых сокетах, т.е. воссозданием работоспособного приложения, либо имитации пакета с помощью, например, *Scapy*. Довольно сложный и специфичный способ разметки трафика.

4. Изменение параметров интерфейса в зависимости от прохождения того или иного приложения через сетевой интерфейс в определенные моменты времени. Такой способ может применяться при условии однозначного и достоверного представления о работающих сервисах в выбранные временные промежутки.

II. Апостериорные виды маркировки трафика - методы, применяемые для разметки уже готовой базы данных трафика:

1. Разметка с использованием специальных инструментов, таких как *PACE* [49], *OpenDPI* [50], *NDPI* [51], *L7-filter* [52], *Libprotoident* [53], *NBAR* [54] или инструментов, работающих на совокупности их результатов [55], является очень популярной среди исследователей и встречается в ряде работ [56; 57]. С технической точки зрения, методы проработаны и выверены на различных трассах, могут применяться в различных условиях, даже при получении базы данных со стороннего источника. Но при подобном методе существует серьезная проблема под названием «*ground truth problem*», что, по сути, является проблемой качества используемого эталона. Кроме того, что выбранные эталонные методы могут содержать некоторые ошибки при идентификации, они все имеют ограниченное количество определяемых классов приложений, что не позволяет исследователям, использующим эти инструменты, выявить какие-либо новые классы. В условиях динамически развивающихся сетей, в особенности, сетей *SDN*, в которых любой участник сети может написать новое приложение, перечисленные методы будут давать большие ошибки, т.к. скорость обновления базы классификации не позволяет в полном объеме учитывать все новые и новые

классы. Более того, перечисленные инструменты могут отделить одно приложение от другого, например, *WhatsApp* и *Telegram*, но не в состоянии разделить между собой мелкозернистые типы услуг (отдельные сервисы), такие как передача сообщений в *Telegram* от голосового вызова в *Telegram*, которые должны иметь различный приоритет по качеству обслуживания. Также перечисленные инструменты хорошо работают на сравнительно небольших базах данных, но при анализе достаточно объемного трафика требуют больших вычислительных мощностей.

2. Ручная разметка при наличии известных использованных видов приложений - трудоемкий процесс, вероятность ошибочной маркировки при котором очень высока.

3. Разметка по файлам регистрации (логов) генераторов трафика, например, *D-ITG*. Метод хорошо подходит при использовании генераторов трафика, но его сложно применить при работе с реальными видами трафика.

I. Разметка с помощью изоляции. Точные и достаточно простые для реализации в моделируемом стенде способы, но практически невозможные при реализации на реальных сетях:

1. Временная изоляция – в определенный момент времени работает только один класс трафика, в другой момент времени работает другой.

2. Физическая изоляция – разделение классов трафика по разным источникам, портам и т.д. Один сервис работает только по одному порту, другой по другому и т.д.

II. Разметка с помощью кластеризации. Кластеризация – способ обучения «без учителя», способный разбить имеющуюся группу объектов по определенным признакам на классы. Т.к. сама по себе является неточной, для верификации других методов обычно не применяется.

#### *1.2.1.3. Сбор статистической информации о потоках трафика*

В случаях, когда база данных трафика поступает не из внешнего источника, важным моментом является выбор метода сбора трафика из сети. Необходимо соблюдать баланс и находить оптимальное соотношение между частотой

обновления информации о потоке, получаемыми параметры потока, и дополнительной нагрузкой на контроллер, сетевые элементы и саму сеть.

Основные способы сбора информации о потоках, применяемые в исследованиях (Рисунок 1.5):

1. Мониторинг моделируемых потоков при помощи файлов регистрации (лог-файлов) генератора. Подход возможен при использовании соответствующих генераторов трафиков. Довольно прост в использовании, не несет дополнительной нагрузки на сетевые элементы, сеть и контроллер, но информация о потоках является очень ограниченной.

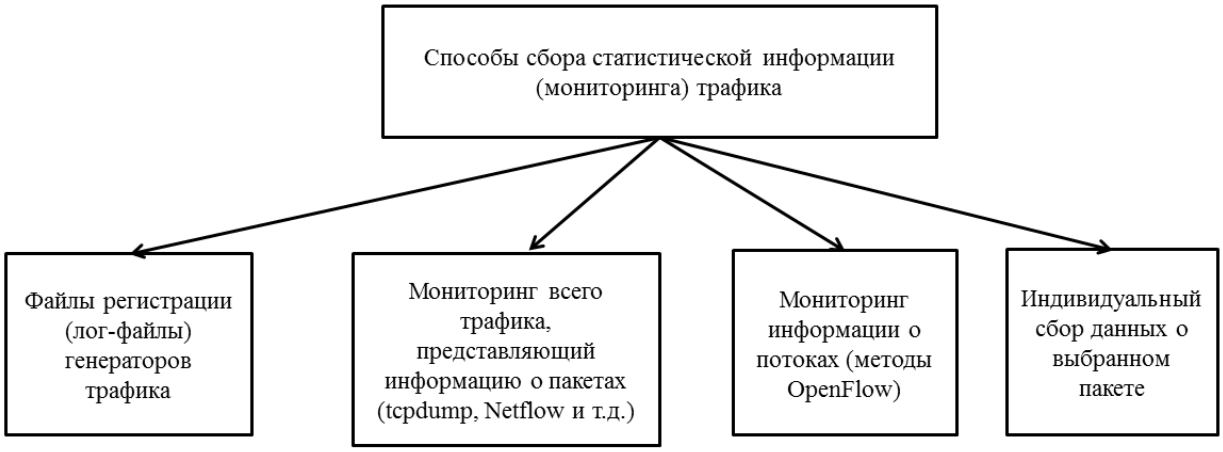



Рисунок 1.5 - Способы сбора статистической информации о потоках трафика

2. Инструменты мониторинга, связанные с получением информации о пакетах. Наиболее популярными являются протоколы *tcpdump* [58], *libpcap* [59], *NetFlow* [60], *SNMP* [61] и т.д. Они дают очень подробную информацию о пакетах, способны группировать их в потоки, работают с различными фильтрами, но, тем не менее, содержат много избыточной информации, которая затрудняет их использование в режиме реального времени, т.к. требует предобработки на узле либо маленькой частоты обновления информации. Кроме того, не все устройства поддерживают те или иные способы мониторинга.

3. Инструменты мониторинга, связанные с получением информации о потоках. Например, встроенные агенты мониторинга контроллеров *ODL* [62; 63] и

*ONOS* [8]. Также применяются различные инструменты, построенные на возможностях протокола *OpenFlow* [64; 65]. Обычно такие средства не вносят больших нагрузок на сеть и позволяют регулировать частоту сбора информации, например, *PayLess* [66], но предлагают ограниченный набор данных по потокам без возможности получения подробной информации о конкретных пакетах, что сужает сферу применяемых с ними методов *ML*.

4. Собственные адаптивные инструменты мониторинга, основанные на возможностях *SDN*-сетей. Развивающиеся *SDN*-сети позволяют не только создавать собственные приложения с использованием *API*-интерфейса, но и задавать логику обработки пакета на коммутаторе (*P4*-сети [67]), благодаря чему открываются неограниченные возможности, связанные со сбором статистической информации о пакетах с любой частотой сбора, любой информацией, в т.ч. временем прохождения определенных этапов обработки коммутатора, информацией о каждом пакете или только об определенных полях определенных пакетов, отсчет которых может проводиться не только с помощью различных фильтров по портам, приложениям, IP-адресам, но и по порядковому номеру пакета или при соблюдении определенных условий, например, при накоплении определенного количества байт с потока [68-70]. Благодаря такой архитектуре, метод также открывает возможности выполнения предварительных расчетов на узле, или нересурсоёмкую проверку по *DPI* для отдельных пакетов, не внося при этом никаких дополнительных нагрузок на сеть, сетевые элементы и контроллер [70].

### 1.2.2. Способы формирования матрицы признаков

Из полученной базы данных формируется матрица признаков, в которой для каждого представленного образца (шаблона) трафика рассчитывается ряд параметров (признаков) и ставится в соответствие маркер класса (Рисунок 1.6).

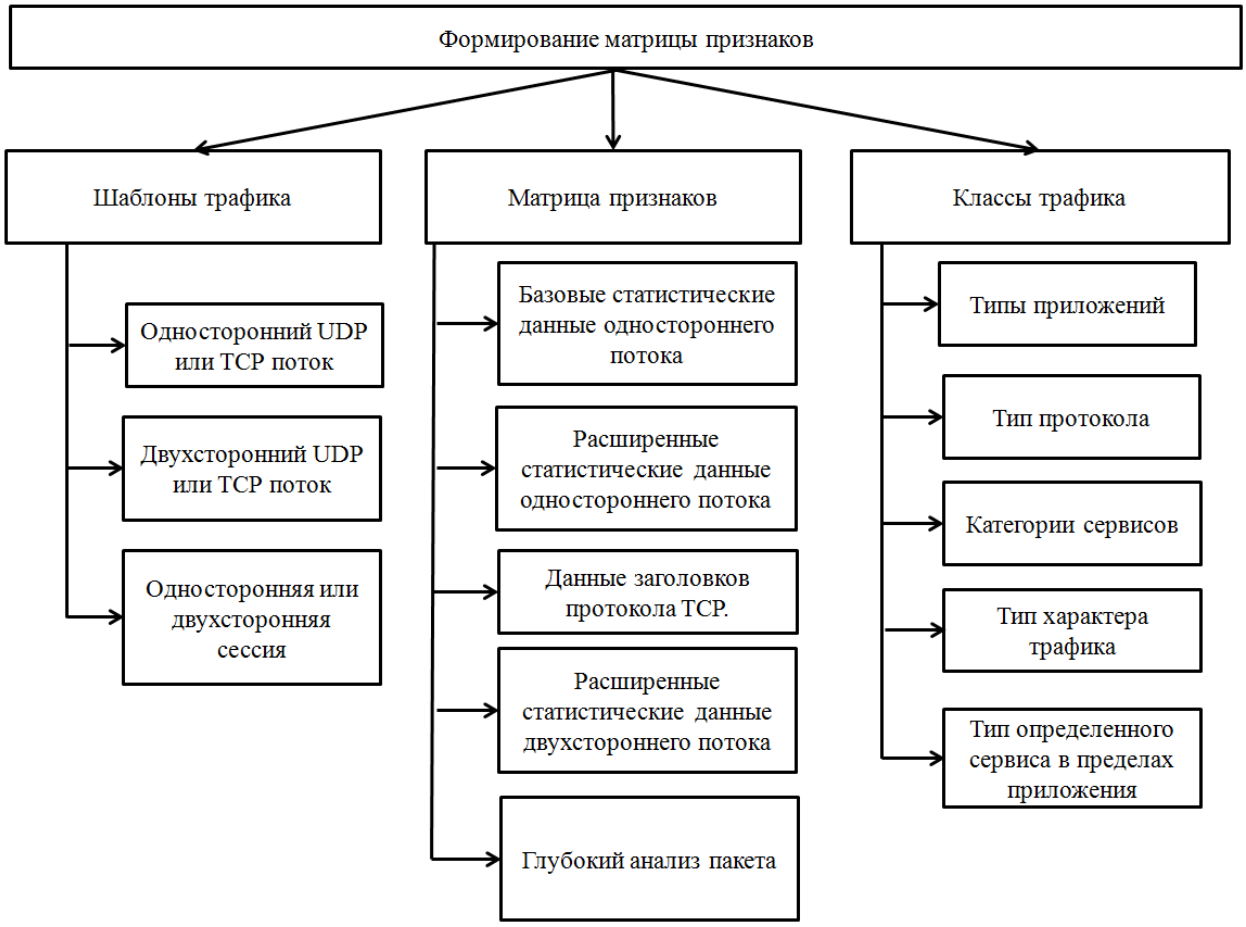



Рисунок 1.6 - Формирование матрицы признаков

#### *1.2.2.1. Шаблоны трафика*

В качества образцов трафика выступают сгруппированные по каким-либо общим свойствам пакеты. Наиболее популярные примеры шаблонов трафика:

1. Односторонний *UDP* или *TCP* - поток, идентифицируемый общими значениями *5-Tuple* - параметров (*IP*-адреса источника и назначения, порты источника и назначения и номер транспортного протокола).

2. Двухсторонний *UDP* или *TCP* - поток, идентифицируемый общими значениями *5-Tuple* – параметров. Рассмотрение в качестве шаблонов двухсторонних потоков с одной стороны, расширяет набор признаков, а с другой, усложняет процесс классификации, т.к. потоки довольно часто бывают ассиметричными как по работе сервиса, так и по используемым сетевым маршрутам.

3. Односторонняя или двухсторонняя сессия, состоящая из разных потоков, образованных с целью создания и поддержания определенного сервиса в рамках одного сеанса. Например, голосовой вызов с помощью приложения *Telegram* сопровождается работой протокола *DNS*, *RTP*, *RTCP* и т.д.

#### *1.2.2.2. Матрица признаков*

Для каждого из образцов трафика составляется вектор признаков (в некоторых статьях – атрибутов), который подбирается в зависимости от того, что принимается за шаблон трафика и каким образом происходит обработка пакетов. Признаки условно можно поделить на следующие категории:

1. Базовые статистические данные одностороннего потока. К ним относятся: время прихода *n*-го пакета на интерфейс и размер *n*-го пакета, записанные для каждого из *n* пакетов, общее количество пакетов. Тип протокола, записанный в поле «порт» источника и назначения *TCP/UDP*-пакета, не относят к признакам, т.к. он может значительно улучшить показатели на статистические характеристики в случаях, когда протоколы соответствуют используемым номерам и позволить допустить множество ошибок при динамически изменяющихся портах.

2. Расширенные статистические данные одностороннего потока. Эти данные получены на основе базовых, путем расчета статистических характеристик, например, среднее время прибытия пакета на интерфейс, средний размер пакета, минимальные и максимальные параметры, СКО, средняя пропускная способность потока на интерфейсе, параметр Херста и т.д.

3. Данные заголовков протокола *TCP*. При исследовании *TCP*-сессий, в качестве атрибутов часто используют такие параметры, как общее количество пакетов с флагами *SYN, ACK, ACK-PSH* (отправленных или полученных), размер окна источника и назначения в пакетах с флагами *SYN* и т.д.

4. Расширенные статистические характеристики двухстороннего потока: направление передачи, отношение количества переданных/полученных пакетов, байт и т.д. Также могут быть использованы различные агрегированные сущности при формировании групп пакетов. Так, например, в работах [56] использовались

«порции» данных, состоящие из двух *TCP*-запросов и одного *ACK*-ответа, а в работе [19] – «раунды», состоящие из всех пакетов на уровне приложений, отправляющихся в одну сторону и подтверждаемые другой стороной.

5. Глубокий анализ пакета – в качестве признаков могут применяться любые данные на уровне приложений, записанные в поле полезной нагрузки и рассчитанные на их основе. Чаще всего трафик в таком случае представляет собой некий «след» или «сигнатуру». Сложность заключается в получении непосредственно самих данных и расширяемости метода применительно к разным протоколам.

#### *1.2.2.3. Основные проблемы, возникающие при обработке данных*

Одним из вопросов, возникающих перед исследователями, является обработка *TCP*-сессий. Протокол *TCP* работает с поддержкой установления соединения и отправляет служебные пакеты (*SYN*, *SYN-ACK*, *ACK*) в начале сессии. Для того чтобы они не оказывали влияние на классификацию, во многих работах информацию о первых трех пакетах *TCP*-сессии убирают из базы данных. Другие же, напротив, используют информацию, получаемую с помощью *TCP*-флагов, и формируют шаблоны трафика на уровне не *TCP*-сессии, а вышележащих протоколов (*HTTP, MSN* и т.д.).

В разных сетях существует разный размер *MTU* (*Maximum Transmission Unit*, максимальная единица передачи), которая в свою очередь влияет на наличие фрагментации трафика. Для того чтобы фрагментация не влияла на результаты классификации трафика, некоторые исследователи убирают данные *TCP* и *UDP*-заголовков, а размеры фрагментированных частей полезной нагрузки соединяют друг с другом.

При формировании потоков может оказаться, что некоторые пакеты потока были потеряны. В таких случаях из базы данных удаляют шаблоны потоков с потерянными пакетами, пренебрегают потерями либо описывают потерянные пакеты по средним характеристикам других пакетов (шаблонов) трафика.

В случаях некачественного сбора базы данных, возможно появление дублирующей информации – обычно повторные записи удаляются из общей базы данных.

При сборе данных на реальной сети можно заметить, что некоторые приложения используются намного чаще других, так, что их потоков в выборке оказывается в тысячи раз больше, чем всех остальных, а некоторые – намного реже и встречаются всего лишь по несколько экземпляров. Редкие по сравнению с другими виды трафика удаляются, т.к. невозможно построить классификатор, обучаясь всего на нескольких примерах. Слишком частые потоки тоже частично удаляются, потому что иначе их резкий количественный перевес может послужить причиной дисбаланса образцов.

Также важно обращать внимание на количество признаков в матрице классификации. Для повышения быстродействия классификаторов рекомендуется сокращать избыточные малоинформативные признаки. Информативность признаков можно определить с помощью инструментов *ML* и способами понижения размерности.

#### *1.2.2.4. Классы трафика*

Наиболее популярные классы, которые используют в качестве результатов классификации с целью поддержки *QoS* в сетях *SDN*:

— типы приложений: *TeamViewer*, *Skype*, *YouTube*, *Gmail* и т.д.;

— тип протокола: *FTP*, *SSH*, *DNS* и т.д.;

— категории сервисов: *WEB*, почта, голос, данные и т.д.;

— тип характера трафика: *Elephant* и *Mice*–потоки (очень большие и очень маленькие потоки соответственно), интерактивный/ неинтерактивный трафик и т.д.;

— тип определенного сервиса в пределах приложения: *Skype*-голос, *Skype*-сообщение и т.д.

### 1.2.3. Методы машинного обучения для определения классов

#### *1.2.3.1. Наиболее распространенные алгоритмы классификации сетевого трафика методами машинного обучения (ML)*

Алгоритмы классификации трафика методами *ML* встречающиеся в современных исследованиях можно разделить на чистые методы *ML* и методы *ML* с применением дополнительных алгоритмов.

В качестве дополнительных методов используются: проверка пакетов по *DPI*, информация, полученная с помощью *DNS*-запросов, проверка по портам общеизвестных протоколов (*SSH, SNMP*) и др. Чаще всего, такие способы позволяют выявить часть общеизвестных потоков и упростить задачу классификации методами *ML*.

Наиболее популярные алгоритмы *ML* могут быть представлены несколькими группами (Рисунок 1.7, [71]):

1. Классическое «обучение с учителем» – наиболее распространенные методы, готовые построить модель по готовым шаблонам трафика. Достаточно точны и просты в использовании. Основной недостаток – сложность внедрения нового потока - для этого требуется получить шаблоны трафика нового потока и переобучить модель. Направления:

— непосредственно классификация (метод «решающих деревьев» (Decision Tree), метод «опорных векторов» (SVM), «Наивный Баейс» (Naive Bayes), k-NN, «логистическая регрессия» (Logistic Regression));

— регрессия («линейная», «полиномиальная», «Лассо»).

2. Классическое «обучение без учителя» позволяет обучать модель даже в отсутствие шаблонов трафика. Основные недостатки: низкая точность алгоритма и необходимость сбора определенной базы данных, на основании которой строятся кластеры. Направления:

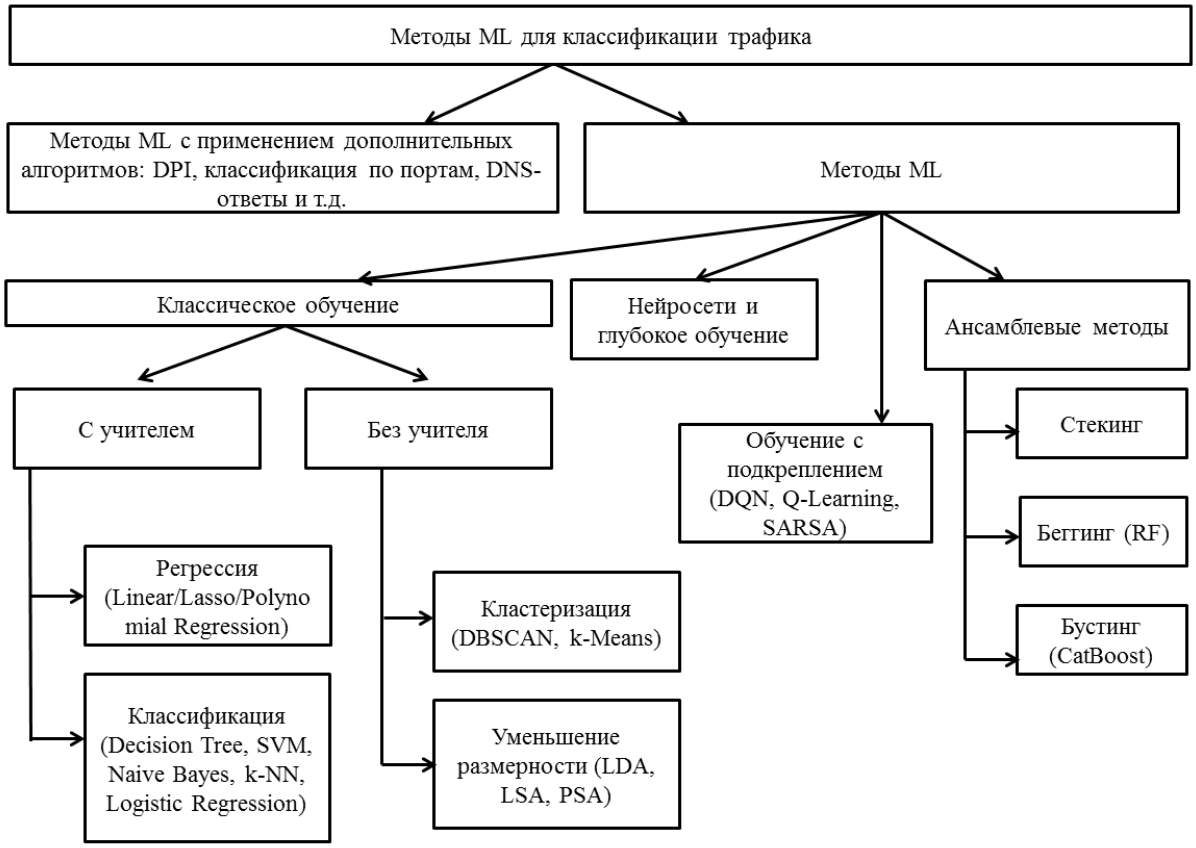



Рисунок 1.7 - Методы машинного обучения для классификации трафика

— кластеризация (*DBSCAN*, *k-Means* и др.) [72];

— уменьшение размерности (*LDA*, *LSA*, *PSA*) позволяет сократить количество признаков.

3. Обучение «с подкреплением» (*Q-Learning*, *Deep Q-Network*, «генетический алгоритм»). Данные для обучения модели не требуются, но требуется подкрепление, т.е. ответ на вопрос, правильно ли классифицировал алгоритм пришедший поток. Достоинством является возможность добавления новых классов.

4. Нейросети и глубокое обучение [73] работают в несколько слоев с заданием весов на каждом уровне. Обладают высокой точностью и позволяют решить даже самые сложные задачи, но требуют больших наборов данных и больших вычислительных мощностей. Примеры: *Multilayer Perceptron* (*MLP*), *Radial Basis Function* (*RBF*), *AutoEncoder Neural Network* (*ANN*), *Convolutional NN* (*CNN*) и т.д.

5. Ансамблевые методы – способы, при которых несколько алгоритмов классических алгоритмов работают вместе. Основные направления:

— «стекинг» – разные алгоритмы обучаются на одной и той же выборке, решение принимает алгоритм, построенный на основе результатов работы предыдущих;

— «беггинг» – один алгоритм обучается на разных выборках, результат вычисляется голосованием [74].

— «бустинг» – один алгоритм обучается последовательно, учитывая ошибки предыдущих алгоритмов. Метод обладает высокой точностью классификации, но не отличается быстродействием.

Примеры ансамблевых методов: *Random Forest* (*RF*), *Stochastic Gradient Boosting* (*SGB*) и *Extreme Gradient Boosting* (*EGB*) и т.д.

#### *1.2.3.2. Инструменты для классификации*

Основные инструменты, применяемые для классификации сетевого трафика:

1. Платформы *WEKA* [75] и *RapidMiner* [76] - интеллектуальные приложения, позволяющие внедрять их в собственные проекты. Отличаются простой использования, достаточно неплохи при проверке какой-то небольшой гипотезы, начальной стадии изучения методов *ML*, но затруднительны при внесении каких-либо значимых изменений в данных, алгоритмах и их комбинациях. Способности исследователя в классификации ограничиваются их возможностями.

2. Языки программирования *R* или более популярный *Python* (библиотека *Scikit-learn* [77]). Отличаются наличием объемных справочных документаций, по сравнению с *WEKA* и *RapidMiner* имеют большое разнообразие алгоритмов, позволяют внедрять алгоритмы непосредственно в проект, т.к. большое количество узлов работает на *Python*, то созданные на нем алгоритмы легко поддерживаются большим количеством устройств. *Python* позволяет взаимодействовать с *Ansible*, потребляет значительно меньше ресурсов и работает

быстрее. Сложность заключается в более высоком пороге вхождения исследователя, т.к. требуется определенное понимание языков программирования.

3. *Statgraphics* [78], *Statistica* [79], *MATLAB* [80] и др. – сильные инструменты для интеллектуального анализа данных, но не работают в режиме реального времени и для больших данных требуют больших вычислительных мощностей. Могут использоваться на этапе разработки метода для проверки каких-либо гипотез или результата работоспособности созданных кодов.

#### *1.2.3.3. Анализ работоспособности алгоритма*

Любой созданный подход, использующий методы *ML*, может оцениваться с двух сторон:

1. Параметры алгоритма, влияющие на скорость работы всей системы. К ним можно отнести:

— среднее количество пакетов потока, необходимое для его классификации;

— среднее время существования потока, необходимое для его классификации;

— скорость построения модели;

— минимальное количество потоков одного класса, необходимое для построения модели;

— возможность дообучения, т.е. добавления новых классов в уже существующую модель;

— сложность внедрения алгоритма в сеть.

2. Оценка результатов работы алгоритма. Для проверки работоспособности разработанных алгоритмов с привлечением «учителя», первоначальную выборку разделяют на две части: обучающую и тестовую. На обучающей выборке строят модель для классификации, а на тестовой – проверяют ее работоспособность. Обычно размеры обучающей выборки намного превышают размеры тестовой. Зачастую стремятся к тому, чтобы шаблоны трафика попадали

в тестовую и обучающую выборку в одинаковых процентных соотношениях среди различных классов. Также могут применяться методы кросс-валидации для независимой проверки работы алгоритма [81].

Также как и в других сферах применения методов *ML*, при классификации трафика может возникать проблема переобучения, т.е. когда модель слишком хорошо работает на обучающей выборке, но плохо на тестовой или на реальной сети. В таких случаях рекомендуют изменить настройки классификатора или же добавить дополнительные ограничения.

Для оценки результата работы алгоритма по тестовой выборке рассчитывается ряд параметров:

— матрица несоответствий (*Confusion matrix*), в которой указываются ошибки классификатора 1-го и 2-го рода;

— доля правильных ответов или ***общая точность*** (*Accuracy*) – доля шаблонов трафика, классифицированных верно;

— ***полнота*** (*Recall*) – доля шаблонов класса от общего числа класса;

— ***точность класса*** (*Precision*) – доля шаблонов класса от числа шаблонов, отмеченных алгоритмом;

— ***F1-мера*** — рассчитывается как среднее гармоническое между ***полнотой*** и ***точностью***.

### 1.2.4. Обзор перспективных исследований по классификации потоков трафика для поддержания качества обслуживания методами машинного обучения в SDN-сетях

На Рисунке 1.8. представлено количество публикаций, посвященных классификации трафика методами *ML*, за 2000-2020 годы в научной базе *Google Scholar* [82]. Графики показывают возрастающий интерес исследователей к этой проблеме. Наиболее популярными оказались методы *Random Forest* и *Decision Tree*, а наименее – *XGBoost*, который был разработан в 2014-м году и еще не получил особого развития в области классификации трафика.

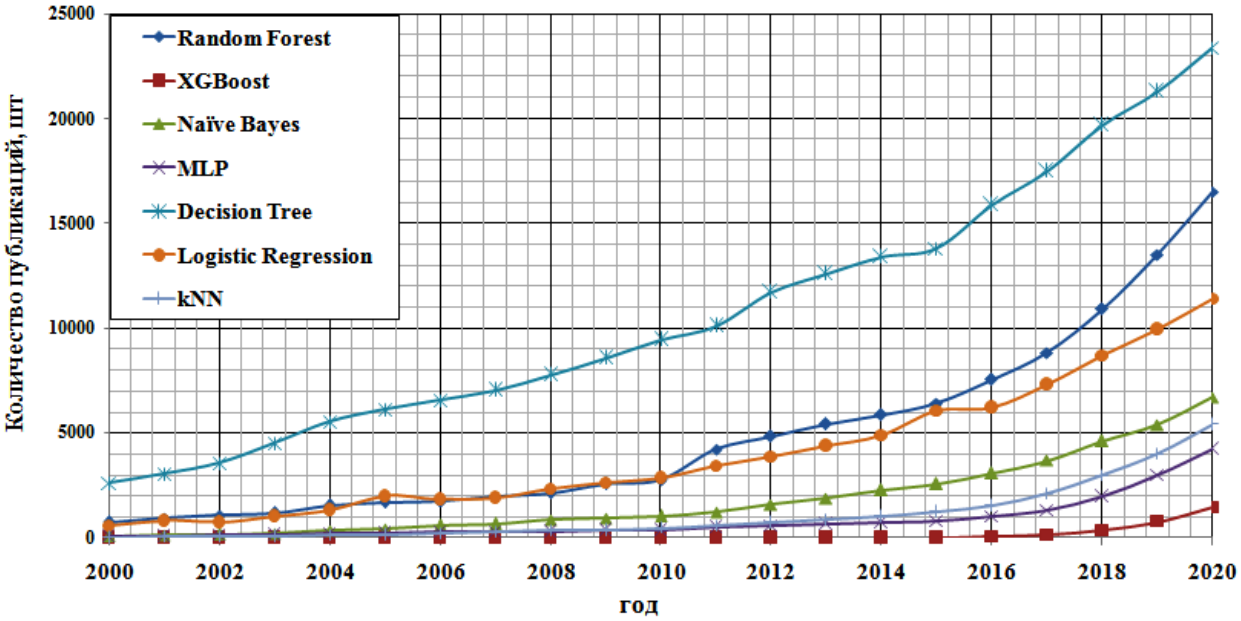



Рисунок 1.8 - Количество публикаций за 2000-2020 года, посвященных классификации трафика методами машинного обучения

В Приложение А вынесена таблица с анализом 20 наиболее перспективных исследований, проводимых в России и за рубежом по классификации потоков трафика для поддержания *QoS* методами *ML* в *SDN*-сетях. Статьи опубликованы в период 2012-2020 и индексируются в международных наукометрических базах данных, таких как *WoS*, *Scopus*, *Google Scholar* и т.д.

Таблицы подробно описывают, какие методы и инструменты используют авторы исследований для решения, описанных в данном исследовании. В описании приводится характеристика работ с трех позиций:

1. Формирование базы данных: способ получения базы данных, ее состав, способы сбора статистических данных и методы их маркировки (Таблица П.1).

2. Формирование матрицы признаков: какие абстракции принимаются за шаблоны трафика, какие признаки являются наиболее важными, по мнению авторов, и что именно в работе подразумевается под классами (Таблица П.2).

3. Классификация трафика: методы классификации, включающие в себя методы *ML* и дополнительные способы, используемые в работе исследователями,

инструменты для интеллектуальной обработки данных и оценка работоспособности алгоритма, по возможности отражающая какое количество пакетов или какая длительность потока необходима для классификации выбранного потока, сколько времени занимает сам процесс обработки данных, а также количество потоков трафика, пакетов в этом потоке или длительность пакета необходимая для построения модели (Таблица П.3). Несмотря на наличие в каждой из работ оценки результатов классификации, в сравнительную таблицу они не вынесены, т.к. сравнение результатов, полученных на различных статистических наборах данных и в различных условиях некорректно. Матрица несоответствий, точность классификации и т.д. являются хорошими показателями для сравнения алгоритмов и подходов, выполненных в рамках одной работы или одних и тех же условиях, но т.к. большинство исследований по ряду причин невозможно в точности повторить, подтвердить или опровергнуть, то они не представляют научного интереса при обзоре статей. Можно лишь отметить, что во всех перечисленных работах результаты классификации оказались очень точными и высокими.

В случаях, когда применяется моделирование *SDN*-сетей, чаще всего применяют сетевой эмулятор *Mininet*, используя в качестве контроллера *ODL, ONOS* или *Floodlight*. В [6] *HP VAN SDN*-контроллер установили на реальной сети, а в [10; 87] для сбора информации о потоках трафика [70] был перепрограммирован непосредственно сам алгоритм обработки пакетов в *P4*-коммуаторах.

Как показано в приложении, во многих случаях авторы статей используют уже готовый набор данных, выложенный в интернете или снятый на реальной сети, и снимают с себя задачи мониторинга потоков и их маркировки, либо используют готовые инструменты типа *Wireshark, tcpdump, tcptrace*. В большинстве случаев даже не даются какие-либо данные о времени построения модели или минимальном количестве пакетов потока, необходимых для его классификации. При этом делаются лишь предположения о работе трафика в режиме реального времени, которые не подтверждаются никакими

исследованиями, при этом принимаются некоторые допущения. Например, в качестве рассматриваемых параметров принимается длительность всего потока, время окончание сессии и т.д., не учитывая, что классификация выбранного потока для поддержания *QoS* совершенно точно теряет свою актуальность, как только такой поток завершится.

Кроме того, в большинстве статей процессом маркировки трафика пренебрегают и результатами работы становятся, в основном, укрупненные категории трафика, полученные с помощью автоматических средств разметки потоков. Это не дает возможности небольшим категориям или приложениям трафика классифицироваться правильно.

Таким образом, большое количество работ в области классификации трафика методами *ML* с поддержкой параметров *QoS* показывает высокую степень заинтересованности научного сообщества в разработке данного исследовательского направления и подтверждает актуальность данной темы. Тем не менее, несмотря на большой проделанный объем работы, как было показано выше, в этой области остается еще много нерешенных вопросов и неучтенных моментов, которые призывают ученых всего мира к дальнейшей разработке данной темы.

## 1.3. Отличия классификации трафика с целью обеспечения качества обслуживания от классификации трафика для целей сетевой безопасности

Классификации трафика с целью обеспечения качества обслуживания (*QoS)* придают особое значение в *SDN*–сетях, т.к. в них чаще обычного появляются новые неизвестные приложения и удается реализовать возможности по управлению трафиком с поддержкой качества обслуживания. В то же время, классификация трафика в целях сетевой безопасности в традиционных сетях применяется довольно давно, и модели классификации на основе машинного обучения регулярно развиваются, прорабатываются и апробируются [83-86].

Такие работы с одной стороны представляют перспективные подходы и решения для классификации трафика, а с другой – требуют повышенного внимания и осторожности при их применении, т.к. между классификацией с целью поддержания *QoS* и с целью обеспечения сетевой безопасности имеются серьезные отличия, как с точки зрения базы данных и матрицы признаков, так и с точки зрения итоговых классов (Таблица 1.1).

Таблица 1.1. Основные отличительные особенности классификации с целью поддержания *QoS* и с целью обеспечения сетевой безопасности

| № | Характеристики | Цель классификации трафика | |
|---|---|---|---|
| | | обеспечение *QoS* | обеспечение сетевой безопасности |
| 1. | База данных | безопасные проверенные потоки | подозрительные, вредоносные и обычные потоки трафика |
| 2. | Классы трафика | приложения, классы *QoS*, категории услуг и т.д. | вредоносный/обычный трафик, наличие/отсутствие атаки |
| 3. | Предварительная обработка данных | удаление/замещение выбросов | выбросы используются для обнаружения вторжений |
| 4. | Матрица признаков | не рекомендуется использовать номера портов; для классификации достаточно статистической информации о пакетах | номера портов анализируются; содержит подробную информацию о прикладном уровне пакетов |

## 1.4. Математическая постановка задачи динамической классификации

Задача классификации трафика может формулироваться следующим образом. Пусть имеется система обслуживания пользовательского трафика

(Рисунок 1.9), на вход которой могут поступать два вида заявок: заявки от размеченных (с помощью меток *DSCP/ToS*, с использованием *VLAN*-тегов и т.д.) потоков: $Y=\{y_1, y_2, \ldots, y_c\}$ и заявки от еще неразмеченных потоков, свойства и природа которых неизвестна: $X=\{x_1, x_2, \ldots, x_k\}$.

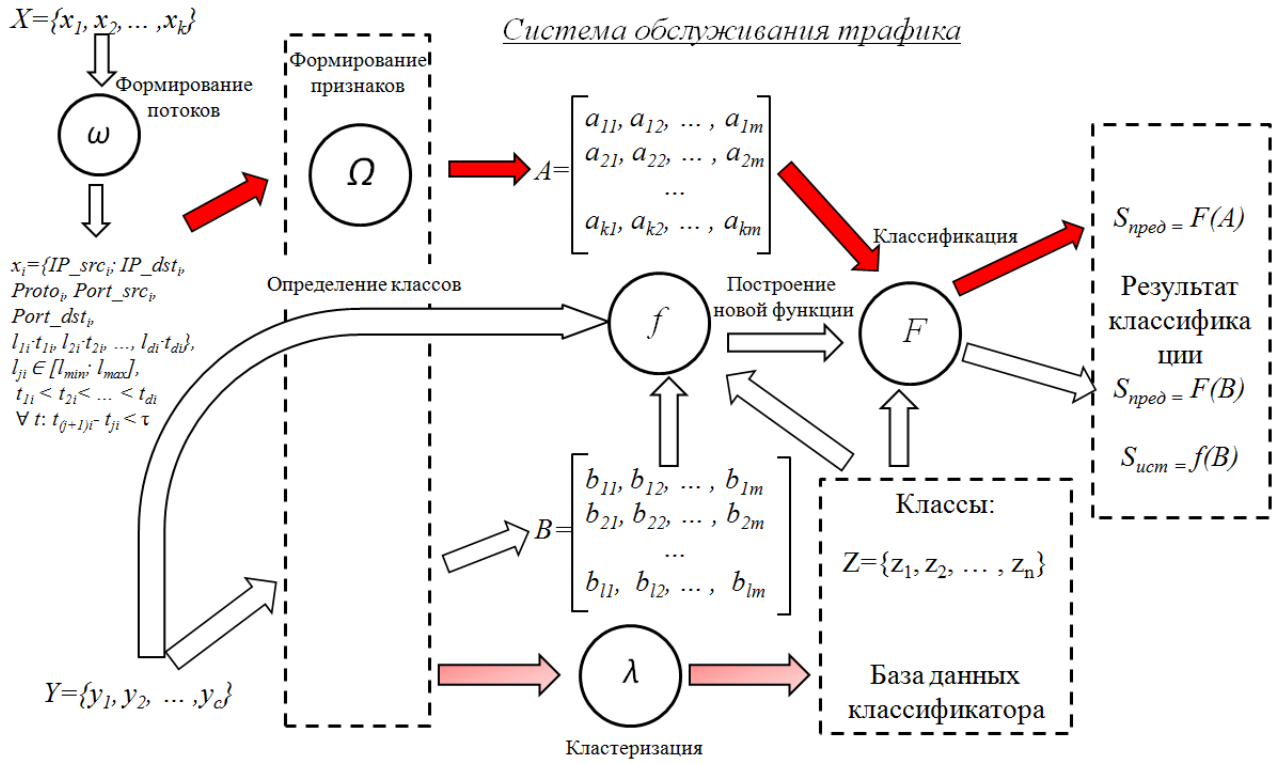



Рисунок 1.9 - Математическая модель классификации трафика для поддержания *QoS* в режиме реального времени

Поток $x_i$ (или $y_i$) является ординарным и представляет собой последовательные, однонаправлено поступающие пакеты и может быть задан вектором: $x_i=\{IP\_src_i; IP\_dst_i, Proto_i, Port\_src_i, Port\_dst_i, l_{1i}\cdot t_{1i}, l_{2i}\cdot t_{2i}, \ldots, l_{di}\cdot t_{di}\}$, где $IP\_src_i$ - *IP*-адрес источника, $IP\_dst_i$ - *IP*-адрес назначения, $Proto_{ix}$ - протокол транспортного уровня, $Port\_src_i$ - порт источника и $Port\_dst_i$ - порт назначения - 5-tuple – идентификаторы потока; $l_{ji} \in [l_{min}; l_{max}]$, $t_{1i} < t_{2i} < \ldots < t_{di}$; – время поступления пакета на интерфейс коммутатора; причем $\forall\ t$: $t_{(j+1)i} - t_{ji} < \tau$ – максимальное межинтервальное время между двумя последовательно поступающими пакетами, определяется из особенностей динамики сети, $l_{min}$; $l_{max}$ –

минимальная и максимальная длина пакета, зависящие от настроек сети и *MTU* (*Maximum Transmission Unit*, максимальная единица передачи). В диссертации исследования проводятся на потоках, в которых $\tau \in (0; 180]$ с, $l \in [40; 100]$ байт.

Каждый из потоков пространства *X* и *Y* с помощью функции Ω может быть описан своим вектором признаков: $\Omega(X)=A=\{A_1,A_2,\dots A_m\}$, $\Omega(Y)=B=\{B_1,B_2,\dots B_m\}$, так что $A_1=\{a_{11}, a_{21}, \dots a_{k1}\}$, $B_1=\{b_{11}, b_{21}, \dots b_{c1}\}$ и т.д., а $x_i=\{a_{i1},a_{i2},\dots a_{im}\}$, где $i=1,2,..k$; $y_j=\{b_{j1}, b_{j2}, b_{j3},\dots b_{jm}\}$, где $j=1,2,\dots,c$. Здесь $m$ – общее число признаков для пространств *X* и *Y*. В качестве признаков используются статистические характеристики потоков трафика.

Под классом трафика подразумевается определенный тип трафика, обусловленный его особенностями: тип приложения, тип сервиса, укрупненные категории и т.д. В модели существует некоторая известная база данных классов трафика - пространство $Z=\{z_1, z_2, \dots,z_n\}$, где $n$– число известных классов. При этом для пространства *Y* существует функция $S_{ист}=f(B)=Z$, однозначно определяющая соответствие между векторами признаков потоков трафика из пространства *Y* и классами из пространства *Z*, а $S_{ист}$ называются «истинными классами». Для режима **обучения** в качестве $S_{ист}$ используют данные о разметке потоков трафика. Также предполагается существование некой функции *F*, такой, что $S_{пред}=F(B)\approx Z$ – предсказанные классы, при этом функция *F* изначально неизвестна и не может быть получена с использованием разметки трафика для пространства *Y*.

На этапе разработки модели, до введения системы обслуживания в эксплуатацию, для оценки работоспособности алгоритма часть потоков трафика из пространства *Y* не отправляется на этап определения функции *F*, вместо этого над ними применяется сама функция *F*: $S_{пред}=F(B)$ и результат сравнивается с $S_{ист}=f(B)$. В терминах машинного обучения этот этап называется **тестированием**.

В результате тестирования для каждого класса подсчитывают количество верно и неверно идентифицированных потоков, составляя матрицу ошибок (Рисунок 1.10). На рисунке количество верно идентифицированных потоков обозначено как $TP_i$, а - неверно $F_{ij}$, причем, i – номер предсказанного класса, а j – номер истинного класса.

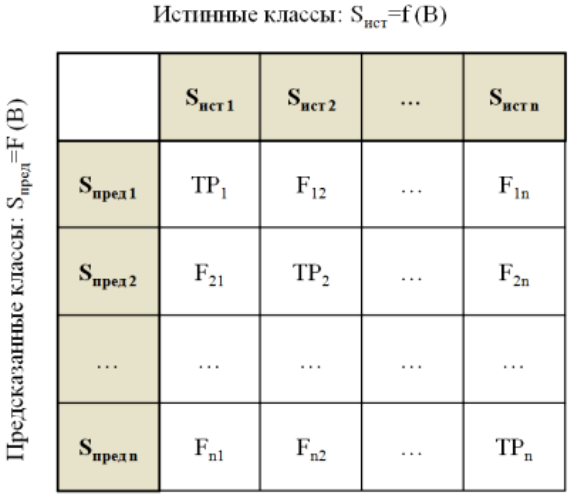



Рисунок 1.10 -Матрица ошибок (Confusion matrix)

По составленной матрице ошибок рассчитываются основные параметры алгоритма:

— ***общая точность*** (*Accuracy*) показывает, сколько было правильно идентифицированных потоков относительно всех потоков:

$$ACCURACY = \frac{\sum_{i=1}^{n} TP_{ii}}{\sum_{i=1}^{n} TP_{ii} + \sum_{i=1}^{n}\sum_{j=1}^{n} F_{ij}}; \quad (1.1)$$

— ***точность*** *i*-го ***класса*** (*Precision*) – это доля верно классифицированных потоков от всех потоков, которым была присвоена метка *i*-го класса:

$$PRECISION_i = \frac{TP_{ii}}{TP_{ii} + \sum_{j}^{n} F_{ij}}; \quad (1.2)$$

— ***полнота*** *j*-го класса (*Recall*) – мера выделения одного потока среди других:

$$RECALL_j = \frac{TP_{ii}}{TP_{ii} + \sum_{i}^{n} F_{ij}}; \quad (1.3)$$

— ***F1-мера*** – гармоническое среднее между точностью и полнотой:

$$F1 = \frac{2 \cdot PRECISION \cdot RECALL}{PRECISION + RECALL}. \quad (1.4)$$

Параметры $ACCURACY$, $PRECISION$, $RECALL$, $F1 \in [0; 1]$, причем, чем выше результат, тем качественнее работа модели.

Для деления данных на обучающую и тестовую выборку используют различные подходы. Одним из наиболее популярных является метод стратифицированной перекрестной *k*-кратной проверки, позволяющий получить результаты классификации независимые от того, на каких данных обучалась сама модель. Всю выборку делят на некоторые «страты» (области) равного размера, затем выполняют классификацию на основе *k-1* области, а одну оставляют для тестирования. Такой эксперимент повторяется *k* раз. Далее рассчитывается ***общая*** средняя ***точность*** на всей выборке.

При добавлении новых классов в модель решается **задача кластеризации методами «обучения без учителя»**. Основными оценками в данном случае являются *согласованный индекс Рэнда* (*ARI, Adjusted Rand Index*) и *гармоническое среднее однородности* и *полноты* кластеров ($V_{measure}$ ). Более подробно в 4.1.3.

Для построения такой модели классификации требуется решить следующие задачи:

1. Разработка матрицы признаков для расчета вектора *A*, т.е. нахождение функции **Ω***(X)*, при этом максимизируются $ACCURACY,\ F1 \to max$.

2. Разработка функции ***F*** с использованием пространства *Y*, *Z* и функции *f*. Функция *F* должна отображать пространство признаков неразмеченных заявок *X* в пространство классов *Z*: $S_{пред}=F(A)\approx Z$, при $ACCURACY,\ F1 \to max$. В терминах машинного обучения – это **задача классификации методами «обучения с учителем»**.

3. Определение базы данных классификатора $Z$ на основе пространств $X$ и $Y$: $\boldsymbol{\lambda}(X, Y) \approx Z$, (при $ARI \rightarrow max$, $V_{measure} \rightarrow max$) или **задача кластеризации методами «обучения без учителя»**.

4. Разработка алгоритма сбора статистических характеристик потоков трафика в *SDN*-сетях для формирования пространств $X$ и $Y$: $\boldsymbol{\omega}(X, Y)$, минимизирующего долю передаваемой служебной информации $\alpha \rightarrow min$.

5. Создание модели классификации трафика с возможностью добавления новых классов на основе композиции разработанных функций: $\boldsymbol{G} = \boldsymbol{\omega} \circ \Omega \circ \boldsymbol{\lambda} \circ \boldsymbol{F}$.

## Выводы по первому разделу

1. Классификация трафика для целей обеспечения *QoS* в *SDN*-сетях является актуальной проблемой и требует особого изучения. В последнее время для решения этой проблемы все чаще применяются методы машинного обучения.

2. Для работы классификатора в режиме реального времени требуется создание специальной матрицы признаков, построенной на информации по первым $n$ пакетам.

3. Классификация в целях сетевой безопасности применяется достаточно давно, многие элементы моделей детально проработаны, но по сравнению с классификацией для обеспечения *QoS*, имеются серьезные отличия на всех этапах работы классификатора.

4. Одной из проблем классификации является возможность обнаружения новых приложений, поэтому требуется создание высокоточного алгоритма кластеризации, способного работать в режиме реального времени.

5. Требуется разработка алгоритма сбора статистической информации о пакетах, адаптированного к созданной матрице признаков.

Результаты работы над первым разделом были представлены на *XI Международной отраслевой научно-технической конференции «Технологии информационного общества»* и опубликованы в [64-65; 88].

# Раздел 2. Формирование матрицы признаков

При разработке математических моделей на основе методов машинного обучения обязательно проводят экспериментальные исследования на моделированных или собранных данных. В данной диссертации решаются вопросы разработки параметров модели, работающей в режиме реального времени, поэтому к базе данных предъявляются следующие требования:

— база данных должна быть достаточно большой (несколько тысяч потоков и более для одного класса), чтобы позволить строить выборки различной величины;

— в базе данных должно содержаться некоторое количество разных типов приложений (6-12), среди которых рекомендуется отбирать как приложения, близкие друг к другу по формату пакетов и интенсивности их поступления, например, протоколы *IMAP* и *IMAPS*, так и непохожие приложения, например, *DNS* и *HTTP*;

— потоки, отобранные для классификации должны быть достаточно длинными (от 10-15 пакетов), в меньших масштабах задача теряет свой смысл;

— потоки должны быть либо промаркированы в самой базе данных, либо формат собранных данных должен быть подлежащим маркировке известными средствами автоматической классификации (*nDPI, L7*–фильтр и т.д.).

## 2.1. Формирование базы данных

Для проведения экспериментов были использованы трассы трафика, полученные на реальной сети исследовательской группой *MAWI* в рамках проекта *WIDE* [89; 90] в контрольной точке *G*, которая представляет собой соединение 10 Гбит/с с *DIX-IE* в Токио, соединяющее США и Японию. Для актуальности собранных данных были выбраны трассы, собранные в 2020 году. Каждая из трасс

представляет собой 15-минутную запись трафика в *pcap*-формате. Информация об использованных наборах данных представлена в Таблице 2.1.

Таблица 2.1. Информация о *pcap*-трассах, используемых в работе

| Название трассы | Время начала захвата | Время окончания захвата | Объем трассировки, Гбит |
|---|---|---|---|
| 202001011400.pcap | 14:00:00; 01.01.2020 | 14:15:00; 01.01.2020 | 19,62215 |
| 202001081400.pcap | 14:00:00; 08.01.2020 | 14:15:00; 08.01.2020 | 40,89192 |
| 202001151400.pcap | 14:00:00; 15.01.2020 | 14:15:00; 15.01.2020 | 37,47634 |
| 202001221400.pcap | 14:00:00; 22.01.2020 | 14:15:00; 22.01.2020 | 33,86615 |
| 202001291400.pcap | 14:00:00; 29.01.2020 | 14:15:00; 29.01.2020 | 34,22953 |
| 202002051400.pcap | 14:00:00; 05.02.2020 | 14:15:00; 05.02.2020 | 34,16715 |
| | | **Итого:** | **200,25324** |

Автоматические анализаторы существующих трасс, такие как *OpenDPI*, *nDPI*, *L7*-фильтр и т.д., предлагают по-возможности объединять потоки в двунаправленные по параметрам *5-tuple* и идентифицировать работающее приложение, с помощью которого они были сгенерированы. Для сбора статистических параметров о потоках из готовых трасс используется как существующий функционал *tcpdump*, так и специально написанные приложения, такие как *ISCXFlowMeter* [91]. Основной проблемой большинства существующих решений является отсутствие возможности получения статистических параметров каждого пакета, вместо которого выделяются средние параметры по всем пакетам, либо отсутствие информации о приложении потока.

Для формирования базы данных из полученных трассировок трафика в работе использовался комплексный подход с применением инструмента *nDPI* для

выделения потоков и идентификации приложений, и модуль *dpkt* языка *Python* для выделения статистической информации о каждом пакете. Так как *nDPI* не всегда справляется с трассами достаточно большого размера, то для предварительного разделения потоков использовалась также утилита *PcapSplitter*, позволяющая делить большие трассы на небольшие потоки и ускорять процесс обработки данных (Рисунок 2.1). В процессе обработки каждый двунаправленный поток делился на два однонаправленных.

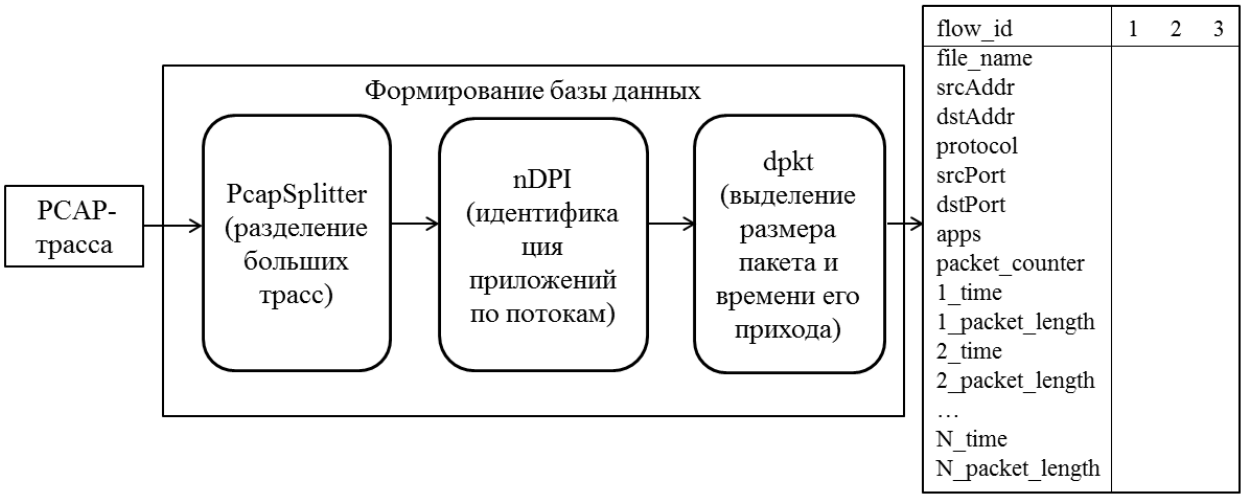



Рисунок 2.1 - Процесс обработки *pcap*-трассы для формирования базы данных

Таким образом, для каждого потока выделяется *flow_id* – присвоенный в результате обработки уникальный идентификатор потока, *file_name* – название файла, с которого был получен данный поток, *srcAddr* – *IP*-адрес источника, *dstAddr* – *IP*-адрес назначения, protocol – название протокола транспортного уровня (в рамках работы отбирались потоки *TCP* и *UDP*), *apps* – название приложения для потока, идентифицированное *nDPI*, *packet_counter* – общее число пакетов в потоке, *time* – время прихода пакета и *packet_length* – размер пакета для каждого из *N* первых рассматриваемых пакетов. Изначально число пакетов выбрано *N=15*, потоки с меньшим числом пакетов были удалены из рассмотрения.

Далее из полученной базы данных удаляются потоки, отправленные в обратном направлении, таким образом, остаются только односторонние потоки. Разность между параметрами *time* для двух последовательно пришедших пакетов образует параметр *time_interval* – время прихода пакетов.

Таблица 2.2. Собранный для анализа набор данных *TCP*-потоков

| № | Название приложения | Количество потоков | № | Название приложения | Количество потоков |
|---|---|---|---|---|---|
| 1 | *DNS* | 111 693 | 21 | *POP3* | 1 299 |
| 2 | *SSL* | 1 864 724 | 22 | *SMB* | 239 |
| 3 | *HTTP* | 225 049 | 23 | *Oscar* | 131 |
| 4 | *Apple* | 102 454 | 24 | *RDP* | 316 |
| 5 | *HTTP_Proxy* | 48 922 | 25 | *LDAP* | 12 |
| 6 | *IMAPS* | 47 525 | 26 | *Oracle* | 1 |
| 7 | *Unknown* | 37 253 | 27 | *Kerberos* | 9 |
| 8 | *SMTP* | 17 357 | 28 | *CiscoVPN* | 15 |
| 9 | *SSH* | 33 180 | 29 | *MsSQL* | 25 |
| 10 | *Skype* | 2 100 | 30 | *H323* | 2 |
| 11 | *CiscoSkinny* | 6 | 31 | *TeamViewer* | 65 |
| 12 | *RTMP* | 4 213 | 32 | *eDonkey* | 1 |
| 13 | *POPS* | 3 859 | 33 | *Redis* | 2 |
| 14 | *Telnet* | 3 985 | 34 | *VNC* | 3 |
| 15 | *IMAP* | 1 699 | 35 | *Tor* | 2 |
| 16 | *Telegram* | 87 | 36 | *TeamSpeak* | 4 |
| 17 | *RSYNC* | 75 | 37 | *SOCKS5* | 3 |
| 18 | *FTP_CONTROL* | 714 | 38 | *Google* | 987 |
| 19 | *OpenVPN* | 18 | 39 | *FTP_DATA* | 11 |
| 20 | *SMTPS* | 527 | 40 | *IAX* | 1 |

В результате обработки выбранных трасс трафика было выделено два набора данных: *TCP* и *UDP*-потоки (Таблица 2.2 и 2.3). Т.к. протоколы *TCP* и *UDP* существенно отличаются друг от друга, режимы работы по ним даже одинаковых приложений также сильно разнятся, и они довольно легко опознаются большинством анализаторов трафика, то принято решение проводить интеллектуальный анализ данных для каждого из этих наборов отдельно.

На основе наборов данных, описанных в Таблицах 2.2 и 2.3 формируются базы данных для последующих экспериментов.

Таблица 2.3. Собранный для анализа набор данных *UDP*-потоков

| № | Название приложения | Количество потоков | № | Название приложения | Количество потоков |
|---|---|---|---|---|---|
| 1 | *NTP* | 39 927 | 12 | *STUN* | 459 |
| 2 | *Quic* | 158 296 | 13 | *UPnP* | 506 |
| 3 | *Unknown* | 20 460 | 14 | *UBNTAC2* | 44 |
| 4 | *SIP* | 82 | 15 | *NetBIOS* | 267 |
| 5 | *IPSec* | 278 | 16 | *NetFlow* | 24 |
| 6 | *DNS* | 119 998 | 17 | *GTP* | 2 |
| 7 | *MDNS* | 16 | 18 | *CiscoVPN* | 3 |
| 8 | *Skype* | 25 | 19 | *DHCP* | 1 |
| 9 | *SNMP* | 470 | 20 | *LLMNR* | 1 |
| 10 | *Apple* | 245 | 21 | *Google* | 7 |
| 11 | *OpenVPN* | 45 | 22 | *VHUA* | 1 |

## 2.2. Формирование матрицы признаков

Матрица признаков формируется на основе статистических данных о пакетах, но подходы к использованию этих данных могут быть различными. В [92] в качестве признаков потока выделяется 248 различных атрибутов. Многие из

этих признаков используются при классификации трафика различными исследователями. В основном применяются только признаки, рассчитанные на основе индивидуальных характеристик потока, а данными по каждому пакету пренебрегают. Например, в [12] используются только минимальное, максимальное, среднее значение, дисперсия и общее число пакетов, среднее число пакетов в секунду, размер пакета, продолжительность потока, а в [11] используют также параметр Херста. В [23] применяют порт сервера, минимальный и максимальный размер TCP-сегмента, отправленного от клиента к серверу, размер окна и количество пакетов с флагом PUSH. В [93] утверждается, что следует избегать временных характеристик пакетов при классификации.

Использование портов источника и назначения в качестве признаков для классификации может привести к ошибочным результатам, т.к. многие приложения используют динамическое изменение портов, шифрование, туннелирование и трансляцию адресов/портов. Вместе с тем, часть приложений работает по фиксированным портам, и в результате важность таких признаков для классификации может оказаться высокой, но давать серьезные ошибки в тех случаях, когда порт не фиксирован.

В условиях проведения классификации в режиме реального времени следует помнить о том, что многие признаки, такие как длительность всего потока, общее число переданных байт, общее число пакетов с различными флагами и др. не могут быть получены до окончания всего потока, т.е. такие признаки не должны использоваться в данном случае. Также, с учетом небольшого количества пакетов, которые могут применяться для классификации, следует избегать таких сложных статистических признаков как параметр Херста, который требует для своего расчета более объемных выборок.

В то же время необходимо создать некоторую матрицу признаков для небольшого числа пакетов ($N=15$), содержащую в себе актуальные, важные и доступные для расчета признаки.

В Таблице 2.4 представлена предлагаемая матрица признаков.

Таблица 2.4. Признаки, рассчитанные на основе статистической информации о каждом пакете

| | № | Название признака | Обозначение | Формула для расчета |
|---|---|---|---|---|
| Набор признаков 1 | 1 | Размер 1-го пакета | 1_packet_length | 1_packet_length |
| | 2 | Размер 2-го пакета | 2_packet_length | 2_packet_length |
| | … | … | … | … |
| | 15 | Размер 15-го пакета | 15_packet_length | 15_packet_length |
| | 16 | Время между приходом 1-го и 2-го пакетов | 1_time_interval | 2_time-1_time |
| | 17 | Время между приходом 2-го и 3-го пакетов | 2_time_interval | 3_time-2_time |
| | … | … | … | … |
| | 29 | Время между приходом 14-го и 15-го пакетов | 14_time_interval | 15_time-14_time |
| Набор признаков 2 | 30 | Средний размер пакета | mean_packet_length | $\frac{\sum_{i=1}^{15} i\_\text{packet\_length}}{15}$ |
| | 31 | Средний интервал времени между пакетами | mean_time_interval | $\frac{\sum_{i=1}^{14} i\_\text{time\_interval}}{14}$ |
| | 32 | Максимальный размер пакета | max_packet_length | $\max_{i\in[1;15]}(i\_\text{packet\_length})$ |
| | 33 | Общее число переданных байт | sum_packet_length | $\sum_{i=1}^{15} i\_\text{packet\_length}$ |
| | 34 | Минимальный размер пакета | min_packet_length | $\min_{i\in[1;15]}(i\_\text{packet\_length})$ |
| | 35 | Медианный размер пакета | median_packet_length | $\underset{i\in[1;15]}{\text{median}}(i\_\text{packet\_length})$ |
| | 36 | СКО размера пакета | std_packet_length | $\sqrt{\frac{15\cdot\sum_{i=1}^{15}(i\_\text{packet\_length}^{\,2})-(\sum_{i=1}^{15} i\_\text{packet\_length})^2}{210}}$ |

| | № | Название признака | Обозначение | Формула для расчета |
|---|---|---|---|---|
| | 37 | Дисперсия размера пакета | var_packet_length | $\frac{15 \cdot \sum_{i=1}^{15}(i\_\text{packet\_length}^{\,2}) - (\sum_{i=1}^{15} i\_\text{packet\_length})^2}{210}$ |
| | 38 | Максимальный интервал времени между поступлением пакетов | max_time_interval | $\max_{i \in [1;14]} (i\_\text{time\_interval})$ |
| | 39 | Минимальный интервал времени между поступлением пакетов | min_time_interval | $\min_{i \in [1;14]} (i\_\text{time\_interval})$ |
| | 40 | Суммарный интервал времени между поступлением пакетов | sum_time_interval | $\sum_{i=1}^{14} i\_\text{time\_interval}$ |
| | 41 | Медианный интервал времени между поступлением пакетов | median_time_interval | $\underset{i \in [1;14]}{\text{median}}(i\_\text{time\_interval})$ |
| | 42 | СКО интервала времени между поступлением пакетов | std_time_interval | $\sqrt{\frac{14 \cdot \sum_{i=1}^{14}(i\_\text{time\_interval}^2) - (\sum_{i=1}^{14} i\_\text{time\_interval})^2}{182}}$ |
| | 43 | Дисперсия интервала времени между поступлением пакетов | var_time_interval | $\frac{14 \cdot \sum_{i=1}^{14}(i\_\text{time\_interval}^2) - (\sum_{i=1}^{14} i\_\text{time\_interval})^2}{182}$ |
| | 44 | Пакетная скорость | packet/s | $\frac{14}{\text{sum\_time\_interval}}$ |
| | 45 | Байтовая скорость | byte/s | $\frac{\text{sum\_packet\_length} - 1\_\text{packet\_length}}{\text{sum\_time\_interval}}$ |

Набор 1 (№1-29) - предлагаемые признаки, которые состоят из индивидуальных характеристик каждого пакета, а набор 2 (№30-45) – наиболее распространенные признаки в исследованиях, рассчитанные на основе набора 1.

## 2.3. Исследование характеристик баз TCP и UDP-потоков

*Набор данных TCP-потоков*

Для того чтобы оценить важность индивидуальных характеристик каждого пакета (№ 1-29) и важность рассчитанных на их основе параметров (№ 30-45) из набора данных *TCP*-потоков (Таблица 2.4) были выбраны следующие приложения: *SSL, HTTP_Proxy, DNS, Apple* (протокол доступа к облачному хранилищу *iCloud*), *IMAPS, HTTP, Skype, SSH, SMTP, RTMP, Telnet, POPS* и *IMAP* как наиболее распространенные в выборке. Инструмент *nDPI* в некоторых случаях может обнаруживать несколько протоколов на разных уровнях, но к приложениям *SSL, HTTP* и *HTTP_Proxy* отнесены все протоколы вышележащего уровня, которые не удалось распознать, поэтому эти классы могут быть неоднородными. Помимо этого, важно понимать, что *nDPI,* как и любой инструмент с подобным функционалом, может допускать ошибки. Кроме того, выбранные приложения охватывают различные услуги сети (голос, видео, данные) и учитывают одни и те же протоколы с разным подходом к безопасности (*IMAP/IMAPS*). Т.к. методы перекрестной проверки и методы машинного обучения рекомендуется применять к сбалансированным выборкам, то для каждого приложения было выбрано по 1500 потоков из имеющихся.

На Рисунках 2.2-2.3 показаны средние значения признаков № 1-29 Таблицы 2.4 для каждого из исследуемых приложений набора данных *TCP*-потоков, а на Рисунках 2.4-2.5 – их диаграммы размахов без учета выбросов. Размер доверительных интервалов при точности 0,95 для диаграммы длин пакетов не превышает 1,5%, а для времени поступления между пакетами - 10%. По построенным графикам видно, что при применении усредненных значений

интервала времени между поступлением пакетов теряется информация об их индивидуальных характеристиках, т.к. они значительно отличаются от средних параметров. В то же время, размер пакетов с 3 по 15 незначительно отличаются друг от друга для ряда приложений (*HTTP_Proxy*, *IMAPS*, *SSH*, *SMTP*, *RTMP*) и имеют видимый размах для остальных приложений (*SSL*, *DNS*, *Apple*, *HTTP*, *Skype*, *POPS*, *IMAP*). При этом размер 1-го и 2-го пакетов для всех приложений значительно отличаются от размеров остальных пакетов.

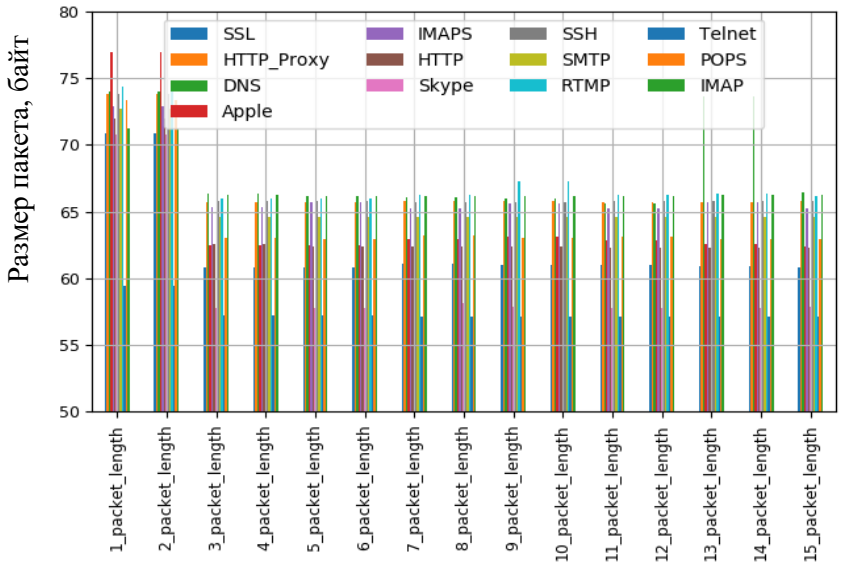



Рисунок 2.2 - Средние размеры пакетов (№ 1-15) для *TCP*–приложений

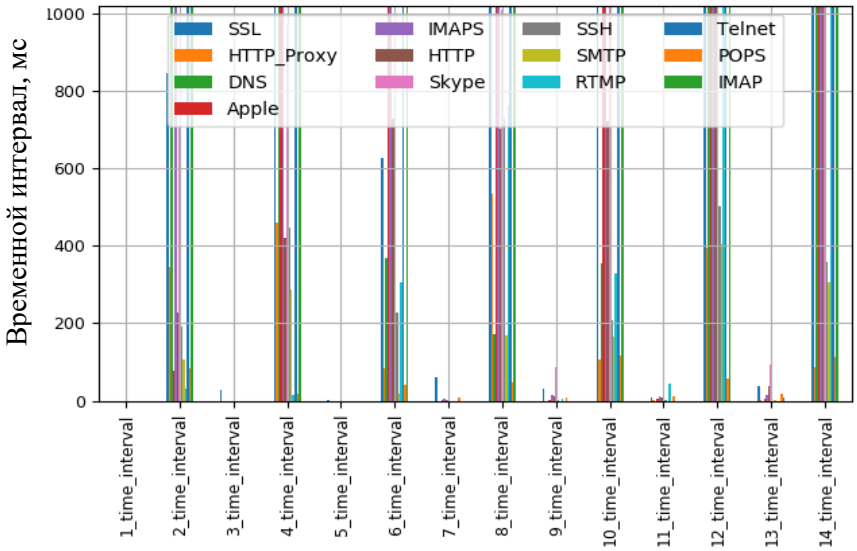



Рисунок 2.3 - Средние размеры интервала времени между поступлением пакетов (№ 16-29) для *TCP*-приложений в интервале от 0 до 1 с

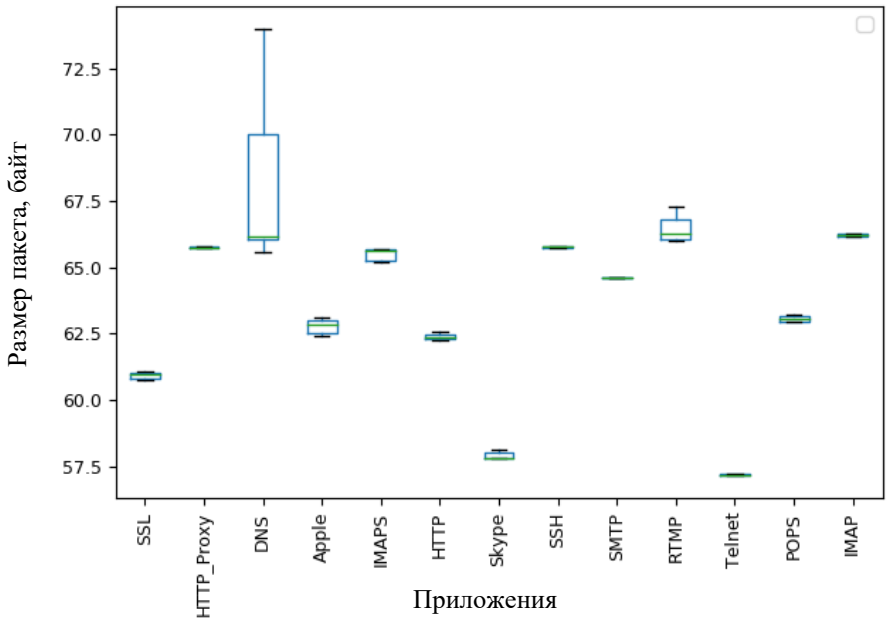



Рисунок 2.4 - Диаграмма размаха размера пакета для TCP–приложений

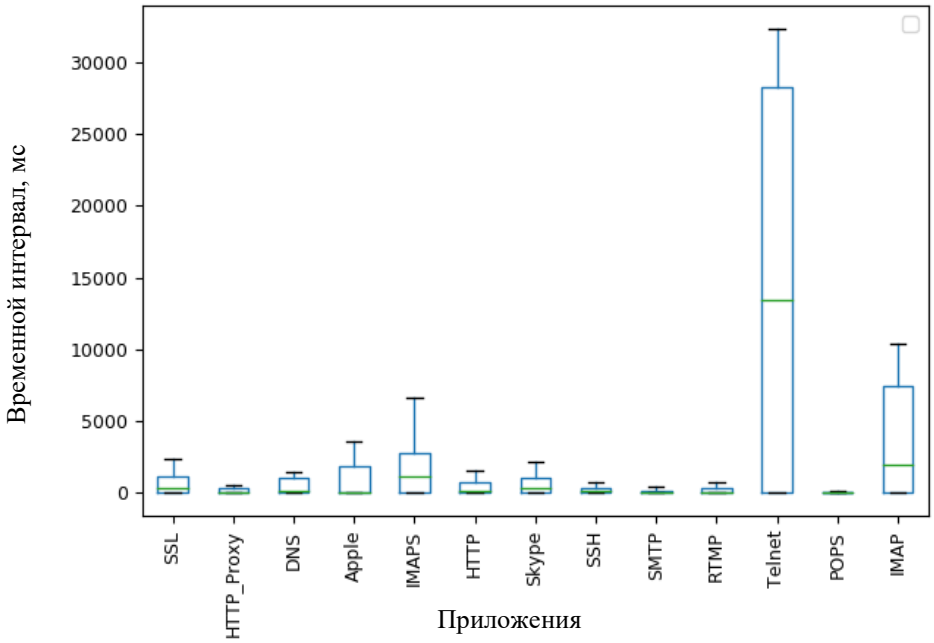



Рисунок 2.5 - Диаграмма размаха интервала времени между поступлением пакетов для TCP–приложений

Диаграмма важности признаков Таблицы 2.4 [94] на основе метода *OOB* (*out-of-bag*) алгоритма *Random Forest*, представленная на Рисунке 2.6, также подтверждает выводы Рисуноков 2.2-2.5 о том, что усреднение характеристик по небольшой выборке в 15 пакетов является недостаточно точной оценкой и приводит к потере информации об индивидуальных характеристиках пакетов.

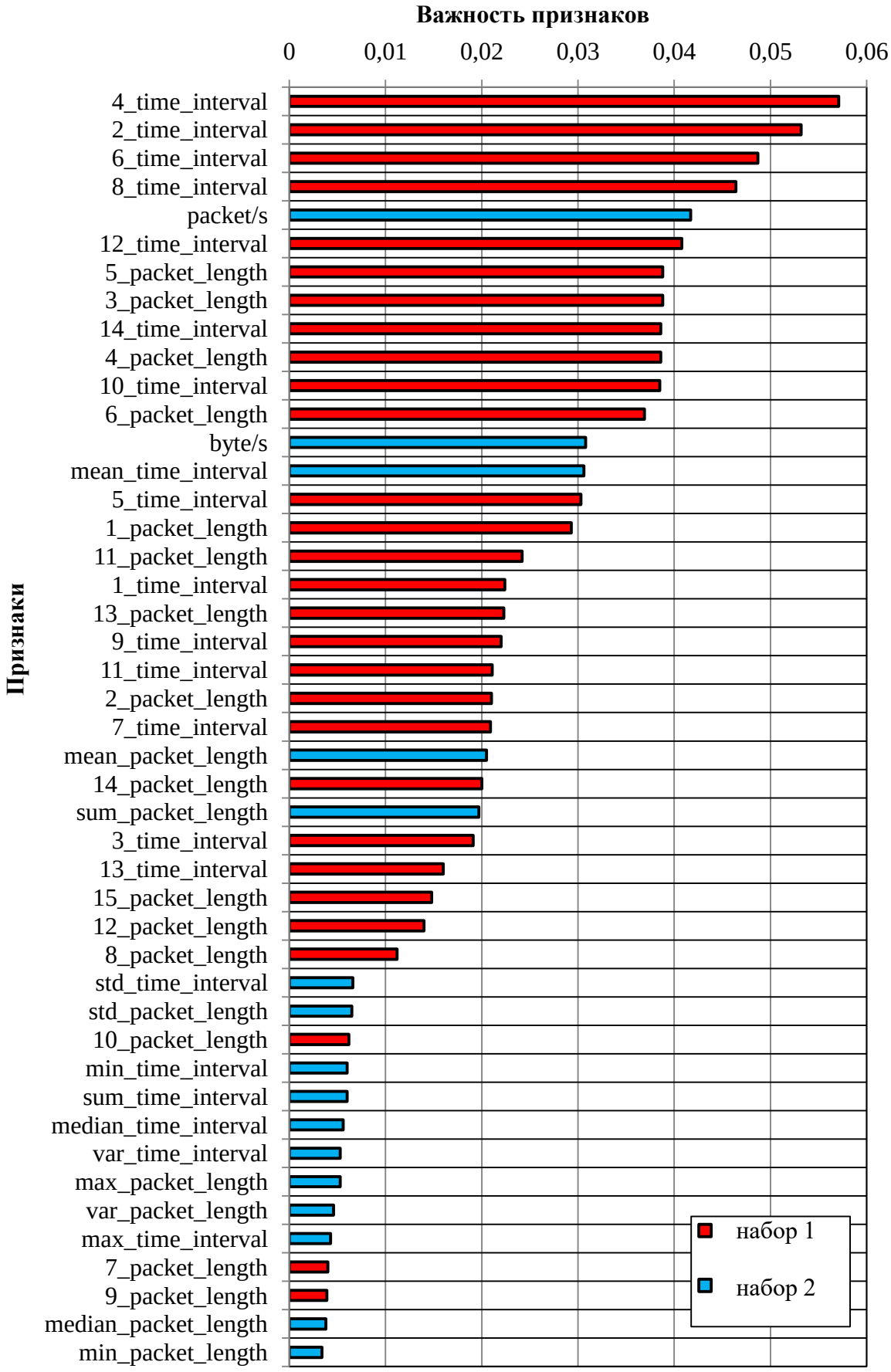



Рисунок 2.6 – Диаграмма важности признаков при классификации *TCP*-потоков

Так, наиболее важными признаками оказались 2, 4, 8, 6-й интервалы времени между поступлением пакетов и пакетная скорость. В целом в числе более важных признаков преобладают индивидуальные характеристики (набор 1) по сравнению с важностью признаков, рассчитанных на их основе (набор 2). Однако характеристики, связанные с размером пакетов, как индивидуальные, так и усредненные, менее значимы по сравнению с временными характеристиками, что опровергает предположение, выдвинутое в [93], о том, что для классификации не нужно использовать временные характеристики.

*Набор данных UDP–потоков*

Для анализа *UDP*-потоков из набора данных Таблицы 2.4 были выбраны следующие приложения: *DNS, NTP, Quic, IPsec, SNMP, NetBIOS, STUN, UPnP* по 250 потоков. Небольшое количество выбранных потоков *UDP* по сравнению с *TCP*-потоками (250 против 1500) объясняется необходимостью получить разнообразное количество приложений (8 приложений по 250 потоков вместо 3-х по 1500), а также особенностью работы методов *ML* на *UDP*-потоках– классификационная модель обучается намного быстрее на *UDP*-потоках, чем на *TCP*. Например, в работе [10] для успешной классификации (близкой к 1 по всем параметрам) *UDP*-потоков было достаточно 60 потоков по 10 пакетов, но эксперимент был ограничен всего пятью приложениями без наличия фонового трафика.

На Рисунке 2.7 изображены средние размеры каждого пакета выбранных приложений. Исходя из полученных данных, можно сделать вывод, что размер пакета существенно меняется для различных приложений и зависит от порядкового номера пакета.

На Рисунке 2.8 изображены средние размеры интервала времени между поступлением пакетов. Из диаграммы видно, что выбранные значения существенно различаются между собой.

Полученные из Рисунков 2.7 и 2.8 выводы подтверждаются диаграммами размаха для размера пакетов (Рисунок 2.9) и интервалов времени между поступлением пакетов (Рисунок 2.10).

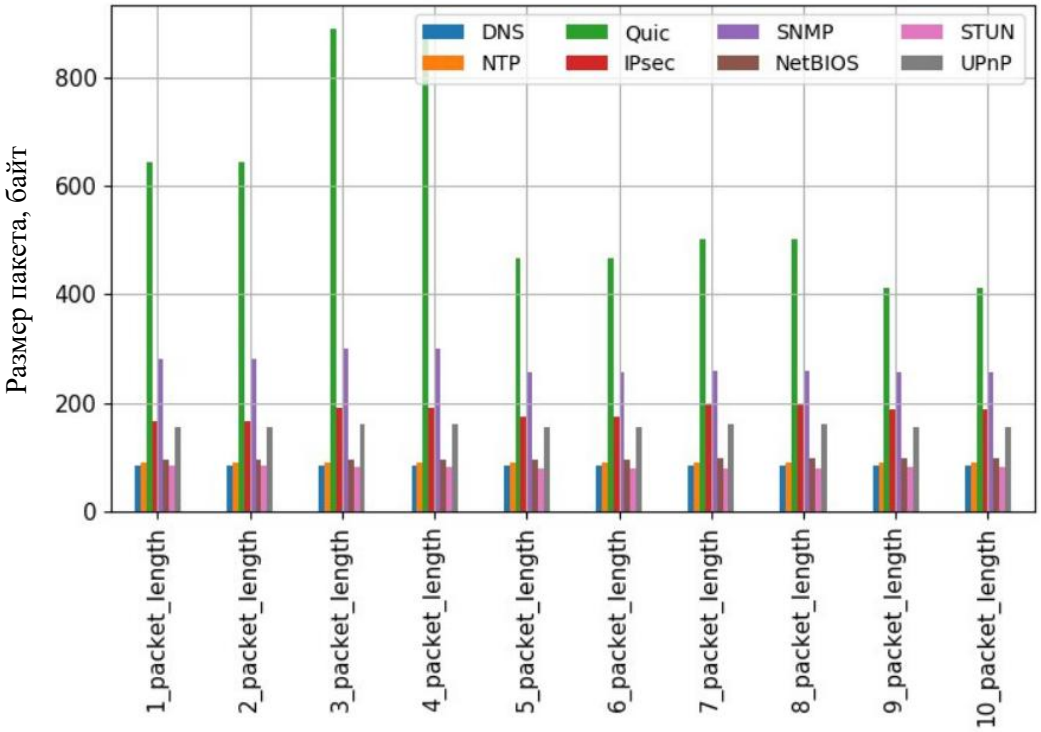



Рисунок 2.7 - Средние размеры пакетов (1-10) *UDP*–приложений

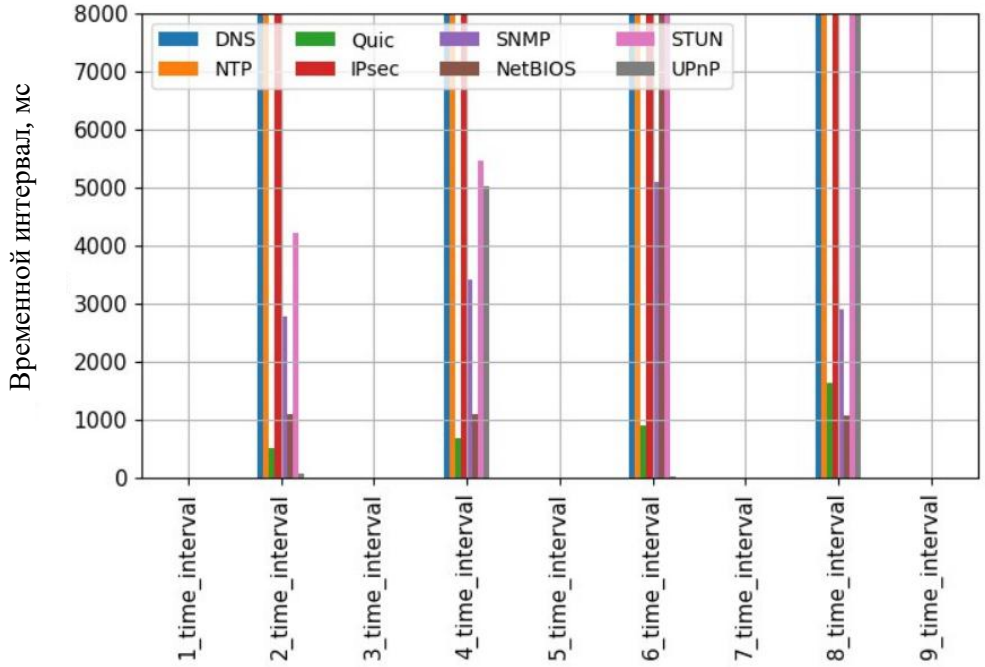



Рисунок 2.8 - Средние размеры интервала времени между поступлением пакетов (16-24) для *UDP*-приложений

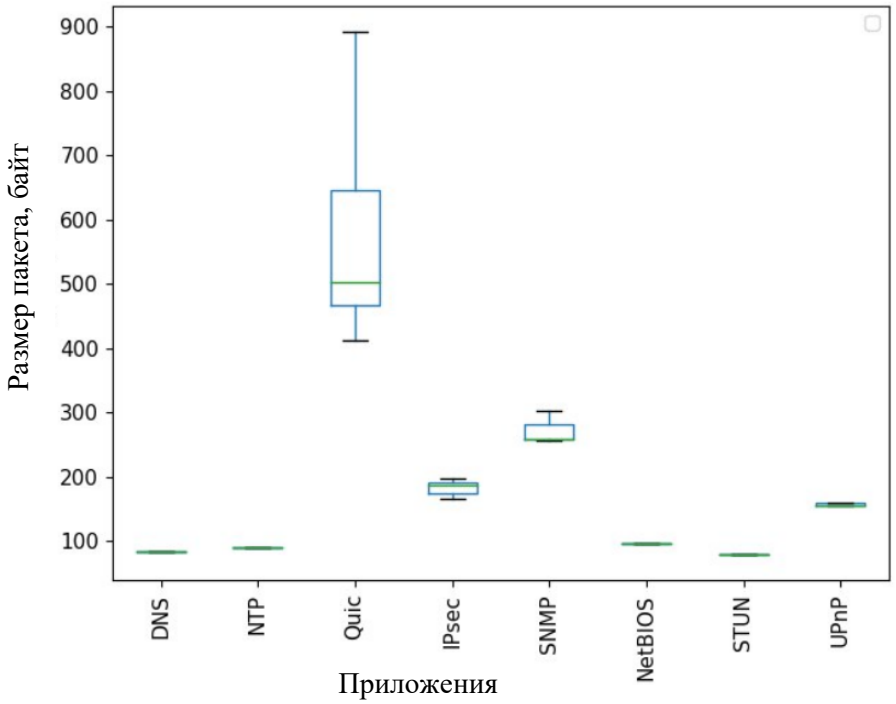



Рисунок 2.9 - Диаграмма размаха размера пакетов для *UDP*–приложений

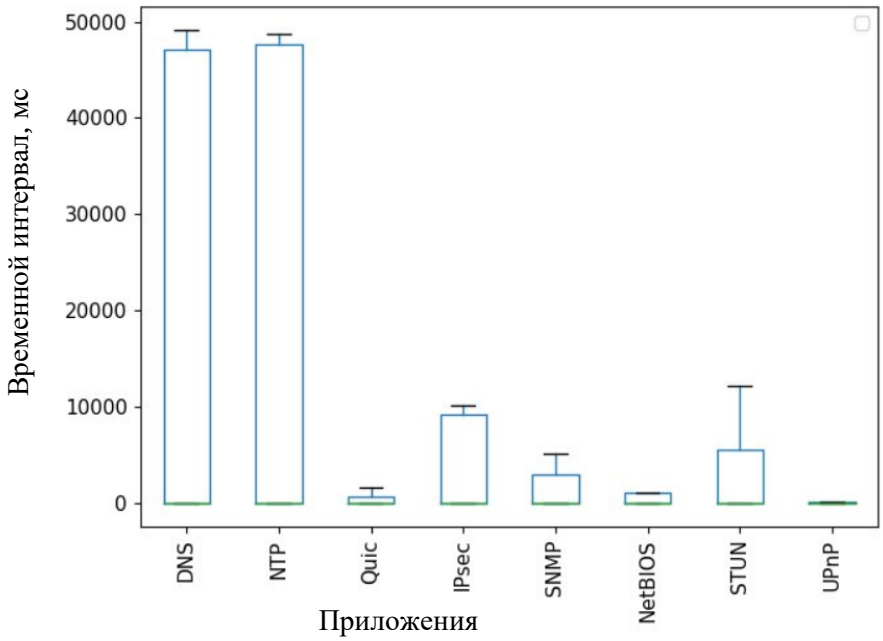



Рисунок 2.10 - Диаграмма размаха интервала времени между поступлением пакетов для *UDP*–приложений

Анализ важности признаков (Рисунок 2.11) показал, что наиболее важными из них являются 3-й, максимальный, медианный размеры пакетов и 2-й, 8-й временной интервал. По сравнению с анализом *TCP*-потоков, для *UDP*-потоков,

рассчитанные характеристики (30-45) также как индивидуальные характеристики, являются достаточно важными признаками.

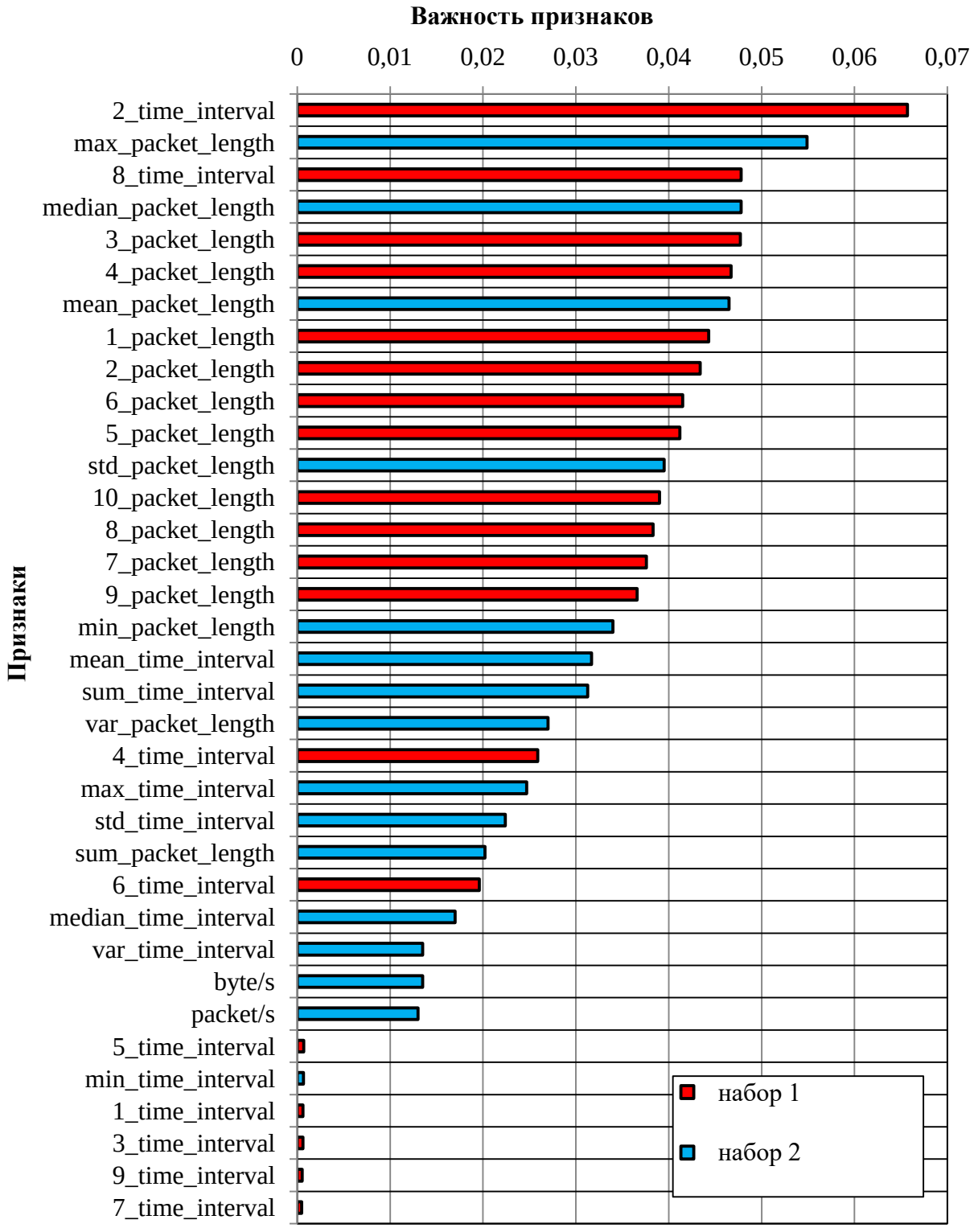



Рисунок 2.11 – Диаграмма важности признаков при классификации *UDP*-потоков

## 2.4. Анализ матрицы признаков на основе результатов классификации

*Классификация TCP-потоков*

Классификация потоков трафика проводилась с помощью библиотеки *scikit-learn* языка *Python* [95]. Выбранный метод машинного обучения – *Random Forest,* (подробно описан в Разделе 3), количество деревьев – 100, остальные параметры выставлены по умолчанию, при разделении потоков использовался метод 5-кратной перекрестной проверки.

В Таблице 2.5 приведены результаты классификации *TCP*-приложений при различных наборах признаков: полная матрица признаков (№ 1-45 Таблицы 2.4), индивидуальные характеристики (набор 1, № 1-29 Таблицы 2.4) и рассчитанные на их основе характеристики (набор 2, № 30-45 Таблицы 2.5). На Рисунках 2.12-2.14 приведена графическая иллюстрация Таблицы 2.5.

Результаты классификации, полученные при использовании индивидуальных характеристик (набор 1, № 1-29), схожи с результатами классификации, полученными при использовании всех признаков № 1-45. Для приложений *SSL, Apple, HTTP, Skype, SSH, SMTP, Telnet, POPS* и *IMAP* результаты классификации при использовании индивидуальных характеристик немного превышают результаты, полученные при использовании всех признаков, а для остальных – немного меньше, но разность незначительная, примерно на 0,001-0,01 доли. Результаты классификации на основе рассчитанных характеристик (набор 2, № 30-45) для всех приложений ниже, чем в двух предыдущих случаев. Разность для большинства приложений значительная, доходит до 0,25 доли (***полнота*** для *SMTP*).

При этом ***общая точность*** классификации для набора 1 на 0,6% выше, чем для полного набора и на 16,9% выше, чем для набора 2.

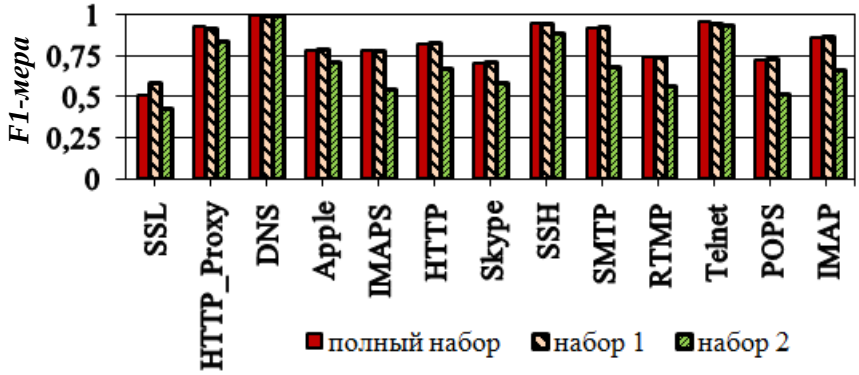



Рисунок 2.12 - ***F1*-мера** классификации *TCP*-приложений при различных наборах признаках

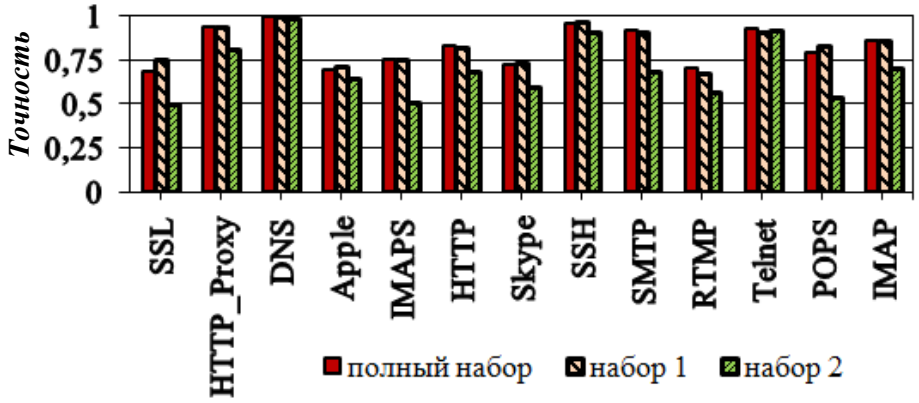



Рисунок 2.13 – ***Точность классов*** при классификации *TCP*-приложений для различных наборов признаков

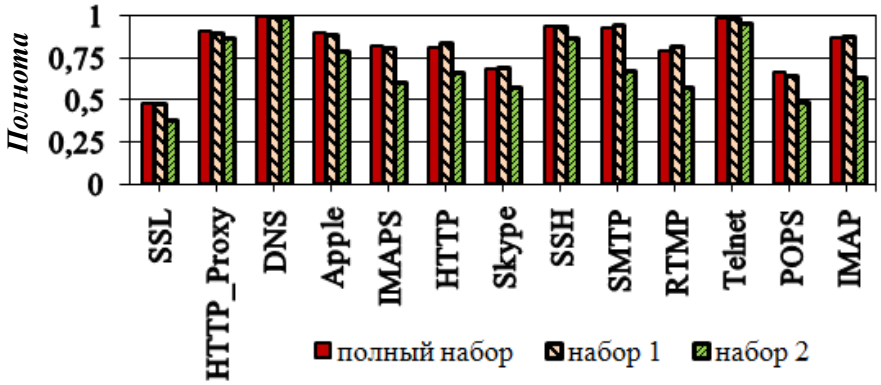



Рисунок 2.14 - ***Полнота*** при классификации *TCP*-приложений для различных наборов признаков

Таблица 2.5. Результаты классификации *TCP*-приложений при различных наборах признаков

| № | Приложение | Полный набор признаков | | | Набор признаков 1 | | | Набор признаков 2 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | ***F1*** | ***Точность класса*** | ***Полнота*** | ***F1*** | ***Точность класса*** | ***Полнота*** | ***F1*** | ***Точность класса*** | ***Полнота*** |
| 1 | *SSL* | 0,507 | 0,681 | 0,477 | 0,589 | 0,755 | 0,483 | 0,430 | 0,496 | 0,380 |
| 2 | *HTTP_Proxy* | 0,927 | 0,942 | 0,913 | 0,917 | 0,934 | 0,900 | 0,837 | 0,810 | 0,867 |
| 3 | *DNS* | 1,000 | 1,000 | 1,000 | 0,998 | 1,000 | 0,997 | 0,995 | 0,990 | 1,000 |
| 4 | *Apple* | 0,780 | 0,690 | 0,897 | 0,794 | 0,715 | 0,893 | 0,712 | 0,648 | 0,790 |
| 5 | *IMAPS* | 0,784 | 0,754 | 0,817 | 0,777 | 0,749 | 0,807 | 0,552 | 0,508 | 0,603 |
| 6 | *HTTP* | 0,820 | 0,827 | 0,813 | 0,828 | 0,820 | 0,837 | 0,670 | 0,680 | 0,660 |
| 7 | *Skype* | 0,702 | 0,726 | 0,680 | 0,713 | 0,731 | 0,697 | 0,586 | 0,599 | 0,573 |
| 8 | *SSH* | 0,949 | 0,962 | 0,937 | 0,951 | 0,969 | 0,933 | 0,890 | 0,907 | 0,873 |
| 9 | *SMTP* | 0,922 | 0,917 | 0,927 | 0,925 | 0,907 | 0,943 | 0,681 | 0,686 | 0,677 |
| 10 | *RTMP* | 0,744 | 0,703 | 0,790 | 0,744 | 0,679 | 0,823 | 0,570 | 0,566 | 0,573 |
| 11 | *Telnet* | 0,960 | 0,931 | 0,990 | 0,947 | 0,913 | 0,983 | 0,937 | 0,914 | 0,960 |
| 12 | *POPS* | 0,721 | 0,790 | 0,663 | 0,729 | 0,830 | 0,650 | 0,514 | 0,540 | 0,490 |
| 13 | *IMAP* | 0,863 | 0,856 | 0,870 | 0,867 | 0,857 | 0,877 | 0,667 | 0,704 | 0,633 |
| ***Общая точность*** | | 0,829 | | | 0,835 | | | 0,696 | | |

На Рисунке 2.15 показано значение ***общей точности*** на основе *OOB* для тестового и обучающего набора данных с учетом СКО для различного количества пакетов. Наилучшие результаты достигаются при исследовании первых 12-15 пакетов.

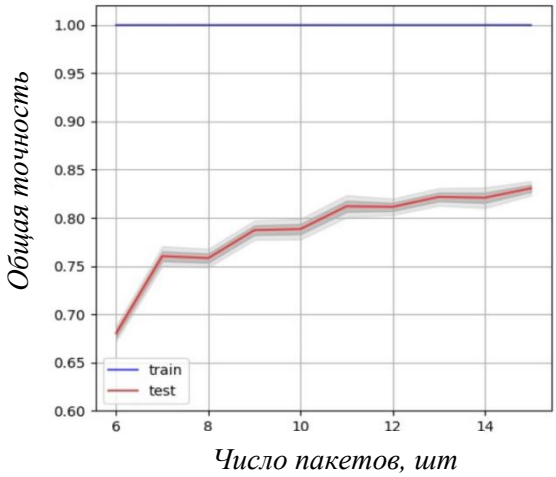



Рисунок 2.15 – ***Общая точность*** *TCP*–потоков при разном количестве пакетов

Таким образом, результаты исследования характеристик набора данных *TCP*-потоков показали, что индивидуальные временные характеристики пакетов оказывают большое влияние на результаты классификации трафика и являются более важными, чем характеристики, рассчитанные на их основе. Основные важные характеристики получены в пределах первых 15 пакетов. В то же время, параметры, связанные с размером пакетов, оказались не так сильно важны по сравнению с временными характеристиками.

*Классификация UDP-потоков*

Результаты классификации *UDP*-потоков трафика при разном составе матрицы признаков методом *Random Forest* представлены в Таблице 2.6. Для приложений *DNS, STUN* и *UPnP* результаты во всех трех случаях (матрица признаков № 1-45, 1-29 и 30-45) получились одинаковыми и близкими к 1. Для приложения *NTP* рассчитанные характеристики показали результаты немного хуже, чем при использовании индивидуальных характеристик и при использовании всех признаков, но разница незначительная, приблизительно на

0,02 доли (для ***точности класса***). Для приложений *Quic, IPsec, NetBIOS* результаты индивидуальных характеристик и рассчитанных характеристик в целом немного хуже (до 0,02 доли), чем при использовании всех признаков. Для *SNMP* классификация, проведенная на основе рассчитанных характеристик, немного выше, чем в остальных случаях.

В целом результаты классификации можно считать приблизительно одинаковыми и достаточно высокими. Для полной матрицы признаков результаты лучше, но в условиях ограниченности ресурсов можно применять также или только индивидуальные характеристики, или только рассчитанные характеристики.

На Рисунке 2.16 изображены результаты ***общей точности*** при разном количестве пакетов для *UDP*-потоков. На графике видно, что наибольшая ***точность*** достигается при использовании 9-10 пакетов и более.

Таким образом, при формировании матрицы признаков *TCP*-потоков очень большое влияние оказывают индивидуальные характеристики пакетов, в особенности временные характеристики и намного меньшее для *UDP*-потоков. Применение их в качестве признаков позволяет улучшить результаты классификаций для некоторых *TCP*-приложений на 20%, а для UDP на 2%.

Минимальным количеством пакетов для успешной классификации потоков трафика оказались 12-15 пакетов набора данных для *TCP* и 10-12 для *UDP*.

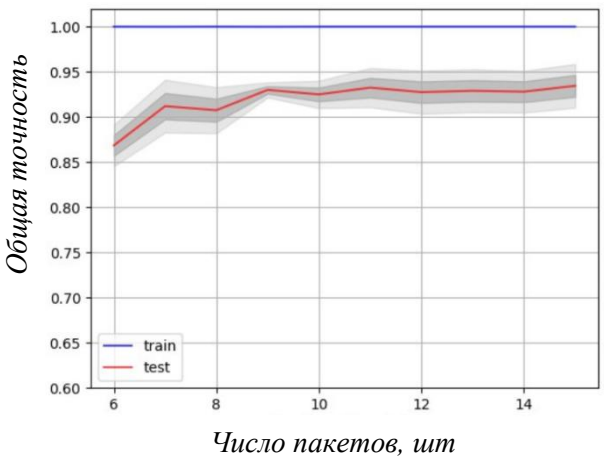



Рисунок 2.16 – ***Общая точность*** *UDP*-потоков при разном количестве пакетов

Таблица 2.6. Результаты классификации *UDP*-приложений при различных наборах признаков

| № | Приложение | Полный набор признаков | | | Набор признаков 1 | | | Набор признаков 2 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | ***F1*** | ***Точность класса*** | ***Полнота*** | ***F1*** | ***Точность класса*** | ***Полнота*** | ***F1*** | ***Точность класса*** | ***Полнота*** |
| 1 | *DNS* | 1,000 | 1,000 | 1,000 | 1,000 | 1,000 | 1,000 | 1,000 | 1,000 | 1,000 |
| 2 | *NTP* | 0,980 | 0,980 | 0,980 | 0,980 | 0,980 | 0,980 | 0,970 | 0,961 | 0,980 |
| 3 | *Quic* | 0,899 | 0,831 | 0,980 | 0,881 | 0,814 | 0,960 | 0,868 | 0,821 | 0,920 |
| 4 | *IPsec* | 0,800 | 0,900 | 0,720 | 0,759 | 0,892 | 0,660 | 0,753 | 0,814 | 0,700 |
| 5 | *SNMP* | 0,949 | 0,959 | 0,940 | 0,909 | 0,918 | 0,900 | 0,960 | 0,960 | 0,960 |
| 6 | *NetBIOS* | 0,980 | 0,980 | 0,980 | 0,961 | 0,942 | 0,980 | 0,980 | 1,000 | 0,960 |
| 7 | *STUN* | 0,928 | 0,957 | 0,900 | 0,939 | 0,958 | 0,920 | 0,928 | 0,957 | 0,900 |
| 8 | *UPnP* | 0,952 | 0,909 | 1,000 | 0,952 | 0,909 | 1,000 | 0,952 | 0,909 | 1,000 |
| **Общая точность** | | 0,940 | | | 0,927 | | | 0,926 | | |

## 2.5. Матрица признаков в сравнении с другими работами

При решении задач классификации трафика существует ряд проблем, затрудняющий сравнение результатов работ различных исследователей между собой. Во-первых, нет единой открытой репрезентативной базы трафика, на которой исследователи могли бы проверить работы, а с учетом темпа развития сетей и услуг, имеющиеся базы быстро теряют свою актуальность. Во-вторых, многие технические моменты, параметры математических моделей классификатора трафика, варианты предварительной обработки данных, способы маркировки базы данных в работах часто не описываются достаточно подробно. В-третьих, при использовании методов *ML* большую роль играют случайные составляющие: деление на тестовую и обучающую выборки, построение моделей (особенно простейших) и т.д. И чем больше и разнообразнее базы данных, тем сильнее влияют перечисленные факторы. Несмотря на это, предпринимается некоторая попытка сопоставить результаты диссертации с результатами других исследователей, сформировав и проклассифицировав своим методом небольшой набор данных с ограниченным набором приложений. Для выбранных потоков формируются также и другие матрицы признаков на основе описаний в работах разных исследователей. По возможности создаются условия, приближенные к оригинальным, но с учетом требования использовать информацию только первых 15 пакетов потока.

В [96] проводится исследование набора данных, полученного в ЦОД в течение 18 часов, общим объемом 315 Гб трафика. В нем выделяются по 10000 потоков для каждого из 6 классов: *DNS, HTTP, RDP, SIP, SSH* и *SSL*. Наиболее важными признаками, выделенными с помощью *RF*, оказались пять признаков: минимальное, максимальное, среднее, дисперсия, СКО полезной нагрузки в пакетов в прямом направлении, такие же пять в обратном направлении, средний интервал времени между поступлением пакетов и его дисперсия. Аналогичное

исследование проводилось в [23] для разных размеров обучающей выборки, начиная от 5000 потоков на один класс.

В [22] для классификации собрали базу данных в университетской сети и выделили в нем 3510 потоков с укрупненных классов: *WWW, E-MAIL, CHAT, P2P, FTP, IM (Instant Message), VoIP*. В качестве признаков выбран набор данных на основе длин пакетов.

В [97] среди 24-х часовых трасс из *Waika to Internet Traffic Archive* отобрали 5 приложений по 2100 потоков в каждом: *FTP-DATA, HTTP, DNS, SMTP* и *HALF-LIFE*. Признаками послужили характеристики: минимальное, максимальное, среднее, СКО размера пакета и времени прихода между пакетами.

В [98] для приложений *HTTP, SMTP, DNS, Socks, IRC, FTP (контроль), FTP (данные), POP3, Limewire* строилась матрица признаков на основе общего количества пакетов, среднего размера пакета и среднего времени прихода между пакетами.

Для сопоставления результатов, было выбрано по 1500 потоков из приложений: *SSL, DNS, HTTP, Skype* и *Telnet*. Матрицы признаков воссоздавались по описаниям из работ, но при этом предварительная обработка данных и настройка гиперпараметров модели не проводились, чтобы создать равные условия для всех работ. Полученные результаты представлены в Таблице 2.7.

Результаты подтверждают, что разработанный в диссертации метод формирования матрицы признаков на основе индивидуальных характеристик пакетов, является более эффективным (на 10-25%) для ранней классификации, в случаях, когда доступна информация только о первых 10-15 пакетах.

Таблица 2.7. ***Общая точность*** классификации при использовании различных матриц признака

| Исследования | эта работа | [96; 93] | [22] | [97] | [98] |
|---|---|---|---|---|---|
| ***Точность*** классификации по методу Random Forest | 0,805 | 0,671 | 0,645 | 0,767 | 0,560 |

## Выводы по второму разделу

В результате работы над вторым разделом был разработан подход к формированию матрицы признаков, обладающий следующими свойствами:

1. Способность работать в режиме реального времени на основе информации о первых 10-15 пакетах потока.
2. Высокая ***точность*** классификации (>80,5%) достигается только за счет применения подобранных признаков, без учета предварительной обработки данных и методов предварительной классификации, что на 10-25% выше результатов на основе других матриц признаков.
3. ***Точность*** классификации при использовании только индивидуальных характеристик для *TCP*-приложений незначительно выше (на 0,6%), чем при использовании совместно с ними рассчитанных признаков, но это позволяет сократить количество признаков с 45 до 29. В то же время, ***точность*** классификации на основе только рассчитанных характеристик ниже на 1,7 % (для *TCP*), что говорит о нецелесообразности такого подхода.
4. Полученная матрица признаков может использоваться для целей обеспечения качества обслуживания.
5. Инвариантность матрицы позволяет применять ее к *TCP* и *UDP*-потокам, а также к зашифрованному и незашифрованному трафику.
6. Признаки для классификации являются легкодоступными для анализа, т.к. основываются на статистических характеристиках пакетов и не требуют глубокого анализа заголовков.

Результаты работы над вторым разделом были представлены на конференции *«Modern Network Technologies (MoNeTec–2020)»,* опубликованы в [94] и использованы в последующих разделах диссертации.

# Раздел 3. Статическая модель классификации трафика

В третьем разделе рассматриваются вопросы построения статической модели классификации трафика, работающей при следующих допущениях:

— существует база данных потоков трафика, проходящего в сети;

— новые приложения в сети появляются редко, а вместе с их введением возникает достаточное количество образцов, для которых имеется однозначное соответствие набор признаков – класс трафика;

— классифицируемые потоки достаточно длинные (>15 пакетов), для других работа модели не имеет смысла.

Данным требованиям полностью отвечают методы машинного «обучения с учителем» (контролируемую классификацию). Предложенная модель статической классификации с их использованием представлена на Рисунке 3.1.

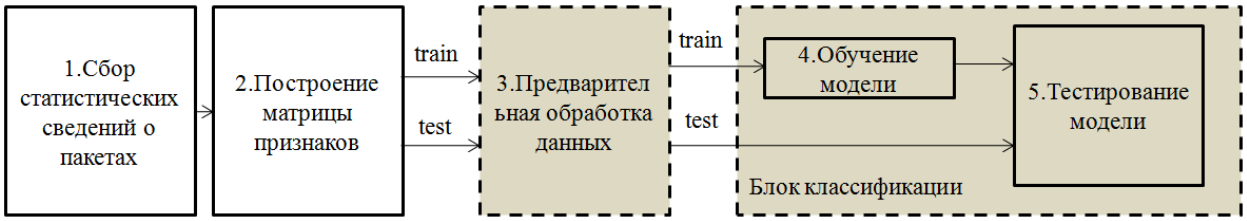



Рисунок 3.1 - Функциональная модель статической классификации трафика

На первом этапе работы модели собираются статистические сведения о пакетах, а на втором - выполняется построение матрицы признаков. Подробный процесс формирования матрицы признаков представлен в Разделе 2, а процесс сбора данных для нее - в Разделе 5. Для экспериментальных целей этой используется база данных, полученная на реальной сети и подробно описанная в Разделе 2.

Для созданной матрицы признаков проводится предварительная обработка данных (третий этап), который включает в себя процессы работы с выбросами, нормализацию, стандартизацию и масштабирование.

В блоке классификации трафика используемый алгоритм машинного обучения проходит процедуру настройки гиперпараметров, которые зависят от его особенностей. Оценка и регулирование выбранного метода проводится с помощью обучающей (*train*) и контролирующей (*test*) последовательностей, которые получаются с помощью стратифицированной перекрестной 5-кратной проверки.

## 3.1. Методы машинного обучения «с учителем»

Среди методов машинного обучения «с учителем» в работах по классификации сетевого трафика можно выделить следующие основные направления: алгоритмы *Decision Tree, Random Forest, Gradient Boosting, Naïve Bayse, Logistic Regression, kNN* и *Neural Network* [99]. Ниже дано их краткое описание и результаты классификации выбранной базы данных с их помощью.

### 3.1.1. Особенности построения классификаторов на основе машинного обучения

#### *3.1.1.1 Алгоритм Decision Tree*

Одним из наиболее распространенных методов *ML* является метод решающих деревьев (*Decision Tree, DT*) [100]. Он позволяет представлять пространство данных в виде разветвленного ациклического графа, элементы которого называют листьями и узлами. В узлах содержится условие разветвления, а в листьях отмечается множество данных, объединенных между собой соблюдением либо несоблюдением выполнения условия, представленного в узле. Лист является концевой вершиной и не имеет дальнейших ответвлений (Рисунок 3.1).

Существует три основных типа *Decision Tree* для задач классификации*: ID3 (Iterative Dichotomiser 3), C4.5/ C5.0* и *CART (Classification and Regression Trees)*. По сравнению с остальными, *CART* способен решать более широкий спектр задач

и принимает разнообразные переменные, поэтому на практике все чаще применяют именно его [101]. При этом строятся бинарные деревья, имеющие на каждом узле по два ответвления.

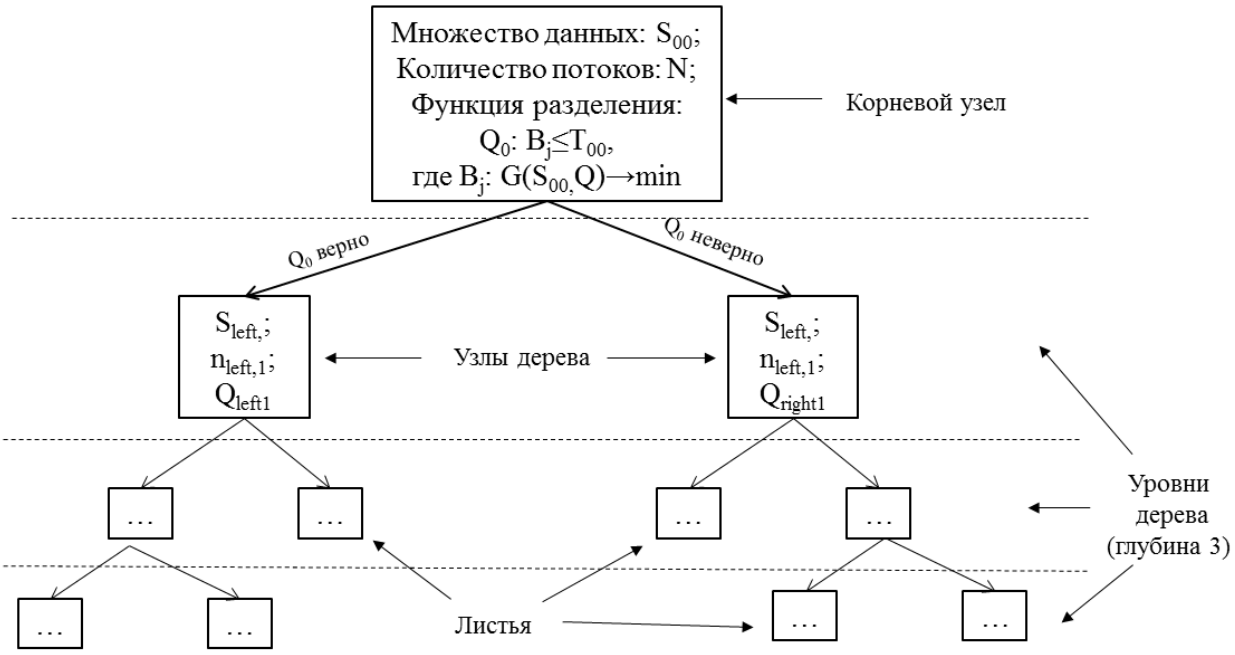



Рисунок 3.1 - Дерево решений, построенное по алгоритму *CART* и его элементы

Алгоритм построения дерева:

1. Пространства обучающей части вектора *B* (вектор признаков размеченных потоков трафика) и *Z* (вектор классов) в соответствии с *S=f(B, Z)* представляются в виде множества $S_{side,i}=(B, Z)$, где *i* обозначает номер уровня дерева и *i =0, 1, 2, ..., max_node*, а *side* обозначает сторону разбиения *side ={left, right, 0}*. Для начального узла *i=0*, *side=0*, т.к. разделений еще не было, и этот узел называется корневым.

2. Для построения дерева $S_{side,i}$ подается на *i*-й узел. Выбирается значение $B_j \in B$ в качестве признака и $T_i$ в качестве порогового значения разделительной функции.

3. Рассчитывается функция *i*-го узла $Q_i=(B_j, T_i)$, разделяющая множество *B* на два подмножества. Определяется левая сторона (*left, 3.1*) и правая (*right, 3.2*):

$$S_{left,(i+1)}(Q_i) = S_{side,i} \mid B_j \leq T_i, \quad (3.1)$$

$$S_{right,(i+1)}(Q_i) = S \backslash S_{left,(i+1)}(Q_i). \quad (3.2)$$

4. Примесь (*impurity*) позволяет количественно оценить качество полученной функции разбиения ($Q$).

Если обозначить за $N_i$ общее количество потоков трафика в $i$-м узле, $n_{left}$ – количество потоков трафика, передаваемые в левый $i+1$ узел, а $n_{right}$ в правый, то для узла $i$ находится примесь (3.3):

$$G(S_{side,i}, Q_i) = \frac{n_{left}}{N_i} H\left(S_{left,(i+1)}(Q_i)\right) + \frac{n_{right}}{N_i} H\left(S_{right,(i+1)}(Q_i)\right), \quad (3.3)$$

исходя из которой выбирается значение $j$ для $B_j$ так, чтобы минимизировать примеси:

$$G(S_{side,i}, Q_i) \to min$$

При $G(S_{side,i}, Q_i) = 0$ в листе остаются потоки, принадлежащие только одному классу.

В качестве функции примеси могут быть использованы различные критерии. Наиболее известными являются: индекс Джини, индекс энтропии и индекс ошибочной классификации.

Пусть частота наблюдений $k$-го класса $Z_k$ в узле $i$ $p_{i,k}$ (3.4):

$$p_{i,k} = \frac{1}{n_i} \sum_{B_i} I(Z_i = Z_k), \quad (3.4)$$

где $I$ – индикаторная функция.

По умолчанию, все веса классов считаются равнозначными.

Индекс Джини показывает, насколько часто поток в узле классифицируется неверно. Функция примеси *H* на основе индекса Джини определяется по формуле (3.5):

$$H\left(S_{side,(i+1)}(\mathrm{Q}_i)\right) = \sum_k p_{i,k}\left(1 - p_{i,k}\right). \quad (3.5)$$

Энтропия показывает наиболее редкие классы в узле и оценивается через логарифм правдоподобия (3.6):

$$H\left(S_{side,(i+1)}(\mathrm{Q}_i)\right) = -\sum_k p_{i,k} log(p_{i,k}). \quad (3.6)$$

Индекс ошибочной классификации (3.7):

$$H\left(S_{side,(i+1)}(\mathrm{Q}_i)\right) = 1 - max(p_{i,k}). \quad (3.7)$$

5. Далее $i$ увеличивается на *1* и пункты 2-4 повторяются для *side={left,right}* пока не выполнится критерий останова.

По умолчанию, деревья решений строятся до тех пор, пока примесь в каждом узле не станет равной *0*, т.е. такой алгоритм не будет содержать никаких ошибок. Такой подход плохо работает на новых данных, так как он является переобученным. В этом случае, нужно использовать какие-то дополнительные критерии остановки, стрижки либо применять ансамбли деревьев. При этом метод стрижки практически не используют, т.к. он довольно сложный и деревья редко строят по отдельности [102].

*Критерии остановки:*

— максимальная глубина дерева (*max_depth*) – количество уровней с узлами, не считая корневого (при *i=0*);

— минимальное количество потоков $B$ для разделения в узле (*min_samples_split*) - по умолчанию, *min_samples_split= 2*;

— минимальное число потоков *B*, находящихся в листе (*min_samples_leaf*) – по умолчанию *min_samples_leaf=1*;

— минимальная доля всех весов от общего числа потоков (*min_weight_fraction_leaf*) в узле – по умолчанию все веса равны друг другу;

— минимальное значение $G\left(S_{side,i}, Q_i\right)$ (*min_impurity_decrease*) – по умолчанию, $G\left(S_{side,i}, Q_i\right) = 0$.

Решающие деревья по отдельности являются слабыми, плохо сбалансированными алгоритмами и успешно справляются только с относительно несложными задачами. Но они могут быть важными компонентами при работе в ансамблевых алгоритмах, поэтому рекомендуется настраивать лучшим образом параметры модели.

Сложность алгоритма *CART* оценивается как время выполнения для построения дерева: $O\ (m \cdot\ l \cdot log(l))$, где $l$ – число потоков для обучения, а $m$ – число признаков и время запроса: $O\ (log(l))$. Для поиска сбалансированного дерева, предполагающего наибольшее снижение примеси $O\ (m \cdot\ l \cdot log(l))$, для каждого узла: $O\ (m^2 \cdot\ l \cdot log(l))$ [101].

#### *3.1.1.2 Ансамбли алгоритмов*

Ансамблем алгоритмов называют совокупность некоторого количества базовых алгоритмов, работающих вместе с целью повышения эффективности модели. Существует несколько основных направлений для реализации ансамблей: «беггинг», «бустинг» и «стекинг» (Рисунок 3.2).

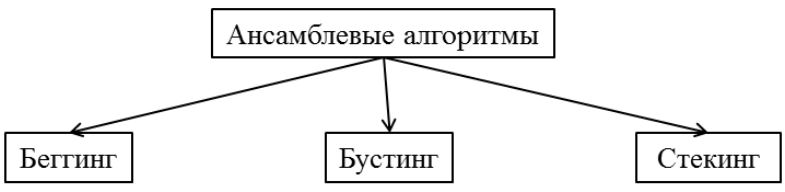



Рисунок 3.2 - Основные направления ансамблевых алгоритмов

В случаях с «беггингом» исходная выборка делится на разные части и один и тот же алгоритм учится на разных выборках. Итоговый результат определяется взвешенным голосованием (Рисунок 3.3), с учетом результатов каждого алгоритма.

При реализации «бустинга» один и тот же алгоритм последовательно учится на своих ошибках. Т.е. исходная выборка подается на какой-то базовый алгоритм, а на основе его ошибок формируется новая выборка, которая снова подается в базовый алгоритм и т.д. (Рисунок 3.4).

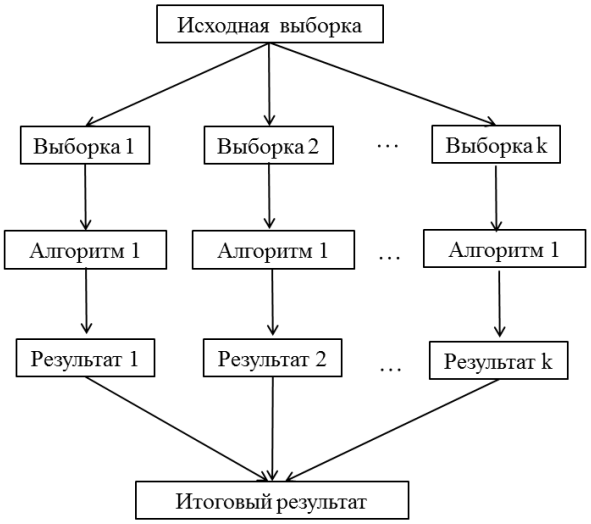



Рисунок 3.3 - Общий алгоритм «беггинга»

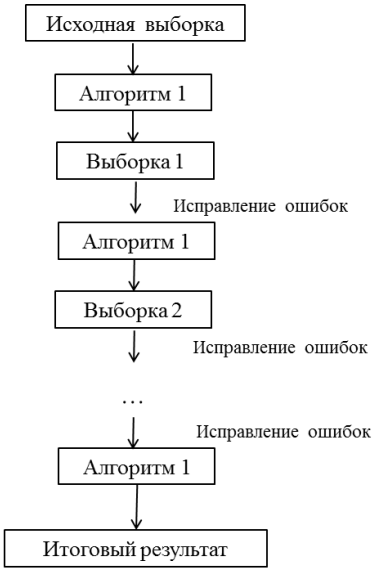



Рисунок 3.4 - Общий алгоритм бустинга

При «стекинге» исходная выборка подается на разные базовые алгоритмы, а результат выводится на итоговый алгоритм, который и принимает окончательное решение по классификации на основании промежуточных результатов каждого алгоритма (Рисунок 3.5).

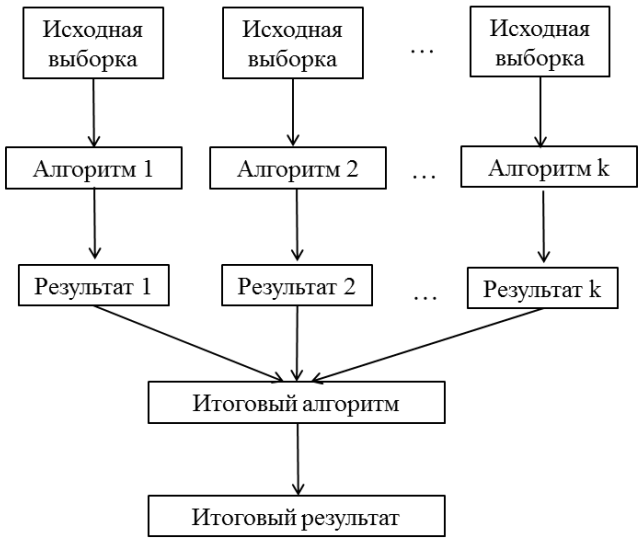



Рисунок 3.5 - Общий алгоритм «стекинга»

### *3.1.1.3 Алгоритм Random Forest*

Алгоритм «случайного леса» (*RF, Random Forest*) [103] представляет собой «беггинг» деревьев Бреймана, т.е. метод, при котором строится множество различных деревьев на разных подпространствах вектора признаков (Рисунок 3.6).

Алгоритм построения «случайного леса»:

1. Обучающее множество $S=f(B, Z, Y)$ разбивается случайным образом на $\{S_{t1},S_{t2}..S_{tn\_estimators}\}\in S$ подмножества, где *n_estimators* означает количество деревьев, участвующих в ансамбле. При этом подмножества $S_t$ могут между собой пересекаться, и в сумме не образуют полное множество $S$. Выборка, не вошедшая в подмножество $S_{ti}$, называется *OOB* (*out-of-bag*) и обозначается $S_{ti,OOB}$. Так как общее число потоков во множестве $S$ равно $l$, а выбор объектов равновероятен, то вероятность попадания потока $S_t$ во множество *OOB* (3.8):

$$P(s_t \in S_{ti,OOB}) = \left(1 - \frac{1}{l}\right)^l. \quad (3.8)$$

При больших размерах множества $S$, т.е. $l \to \infty$ (3.9):

$$\lim_{l\to\infty} P(s_t \in S_{ti,OOB}) = \lim_{l\to\infty}(1 - \frac{1}{l})^l = \frac{1}{e} \approx 0{,}37. \quad (3.9)$$

Таким образом, при больших размерах обучающей выборки, примерно 37% потоков не попадут в подмножества $S_t$ и не будут использованы для обучения.

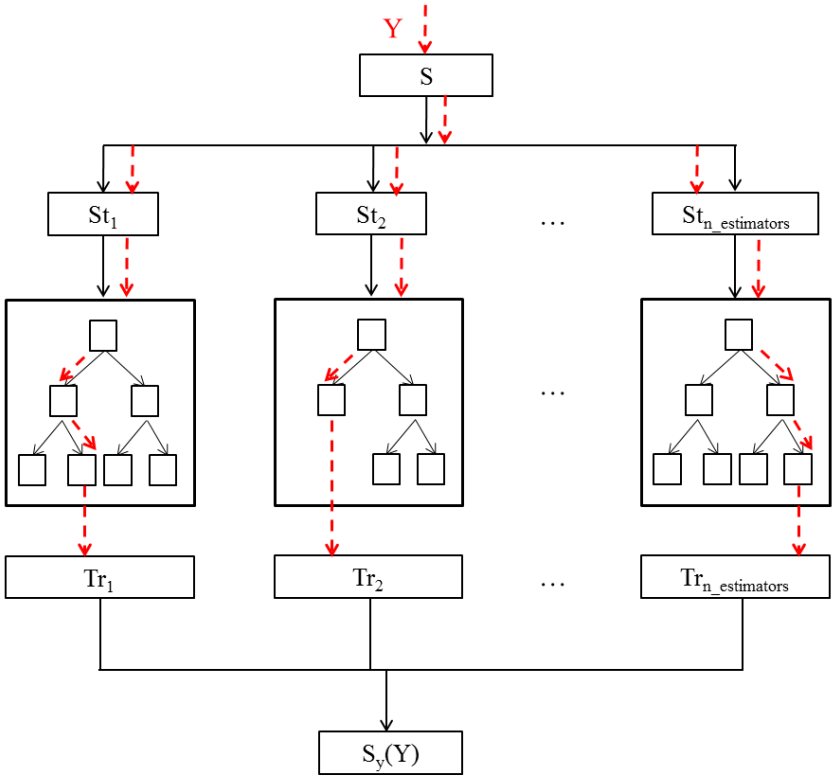



Рисунок 3.6 - Классификация сетевого трафика по алгоритму *Random Forest*

2. На основании каждого из подмножеств $S_t$ независимо друг от друга строится по одному дереву. Построение отдельных деревьев проходит по тем же принципам, что и построение по алгоритму *CART*, но имеет две специфические особенности. Во-первых, в качестве просмотренных признаков в каждом узле по

умолчанию ставится значение: *max_features*=$\sqrt{m}$. Во-вторых, по умолчанию деревья строятся полными, без стрижки.

3. Для каждого отдельного потока $y_j$ каждое из деревьев дает свой прогноз $Tr_i$ *(St$_i$)*, где $i=1,2, \ldots, n\_estimators$. Итоговое решение по классификации определяется как усредненное значение между результатами отдельных деревьев (3.10):

$$S_y(y) = \frac{1}{\text{n\_estimators}} \sum_{i=1}^{\text{n\_estimators}} Tr_i(y). \qquad (3.10)$$

Сложность *RF* определяется как $O\ (n\_estimators \cdot\ l \cdot log(l))$.

*Оценка качества алгоритма RF*

Для оценки качества алгоритма *RF* помимо стандартных параметров, полученных с помощью матрицы ошибок, также применяются параметры, рассчитанные с помощью метода *OOB*. Метод *OOB* заключается в построении «случайного леса» на основе потоков, не участвующих в обучении выбранных для тестирования деревьев и последующем подсчете количества ошибочке классификации по *OOB* ($ER_{OOB}$, 3.11):

$$\forall\ \mathrm{s_t} \in \mathrm{S_{t,OOB}}:\ \ ER_{OOB} = \frac{1}{N_{OOB}} \sum_{i=1}^{N_{OOB}} I(\mathrm{Sy(Y)} \neq \mathrm{f(S,Y)}), \qquad (3.11)$$

где $N_{OOB}$ – размер множества $\mathrm{S_{t,OOB.}}$

### *3.1.1.4 Алгоритм XGBoost*

Алгоритм экстремального градиентного бустинга (*XGBoost*, *XGB* или *EXtreme Gradient Boosting* [104]) представляет собой бустинг деревьев. Общий алгоритм *XGBoost* выглядит следующим образом:

1. Предсказание ансамбля алгоритмов ($\widehat{y}_i$) на основе деревьев имеет вид (3.12):

$$\widehat{y}_i = \sum_{k=1}^{\text{n\_estimators}} f_k(x_i), \quad f_k \epsilon \mathcal{F}, \tag{3.12}$$

где $n\_estimators$ – количество деревьев;

$f_k$ – базовые алгоритмы;

$\mathcal{F}$ - функциональное пространство всех возможных деревьев модели.

2. После получения предсказания от модели, оптимизируется целевая функция ($obj$) или функция оптимизации бустинга, которая имеет вид (3.13):

$$obj(\theta) = L(\theta) + \Omega(\theta), \tag{3.13}$$

где $\theta$ – шаг модели;

$L(\theta) = \sum_{i=1}^{n} L\left(y_i, \widehat{y}_i^{(t)}\right)$ –специфичная функция потерь (может оцениваться как СКО);

$y_i$ – значение $i$-го элемента обучающей выборки;

$\widehat{y}_i^{(t)}$–предсказание для первых $t$ деревьев;

$\Omega(\theta) = \sum_{i=1}^{t} \Omega(f_i)$ -регуляризатор для каждого из деревьев (3.14):

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \mathrm{w}_j^2, \tag{3.14}$$

где $T$ – число листьев;

$\gamma$–параметр, позволяющий регулярировать деление листа на поддеревья;

$\lambda$ – параметр регуляризации для суммы весов деревьев;

$\mathrm{w}_j^2$ – вес листьев;

$\frac{1}{2}\lambda\sum_{j=1}^{T}\mathrm{w}_j^2$ – функция, контролирующая сумму весов модели.

В отличие от *CART*, при котором деревья либо урезаются после их полного построения, либо вводятся ограничения на глубину дерева/ размер листьев/ вершины, при *XGBoost* более гибкая система регуляризации. При построении дерева используется функция, на основании которой принимается решение целесообразности о разделении листа на ветки (3.15):

$$Gain = \frac{1}{2}[H_L(\lambda) + H_R(\lambda) - H_{\text{исх}}(\lambda)] - \gamma, \qquad (3.15)$$

где $H_L(\lambda)$– информационное усиление на новом левом листе;

$H_R(\lambda)$–информационное усиление на новом правом листе;

$H_{\text{исх}}(\lambda)$ - информационное усиление на исходном листе.

Если $Gain < 0$, то деление листа на ветки не происходит, и лист становится терминальным.

3. В модель добавляется новое дерево $f_k$, которое строится на основании ошибок предыдущего шага [105].

При использовании различных функций потерь получаются различные виды «бустинга» (*AdaBoost* при экспоненциальной, *LogitBoost* при логарифмической, *GentleBoost* и др.). «Градиентный бустинг» является обобщением всех этих функций, а «экстремальный градиентный бустинг» позволяет проводить более гибкую регуляризацию, благодаря чему, *XGBoost* достигает больших результатов по сравнению с *AdaBoost* и прочих подобных методов и применяется значительно чаще в современных задачах.

#### *3.1.1.5 Алгоритм Gaussian Naive Bayes*

Алгоритм «Гауссовского Наивного Байеса» (*GNB, Gaussian Naïve Bayes* [106]) основан на применении теоремы Байеса, в котором делается «наивным» предположение об условной независимости между каждой парой признаков.

Несмотря на то, что предположение о независимости в большинстве случаев неверно, *GNB* часто дает достаточно высокие результаты.

Правило классификации для *GNB* имеет вид (3.16):

$$\hat{z} = \arg\max_{z} P(z) \prod_{i=1}^{m} P(x_i | z), \tag{3.16}$$

где $P(z)$ – вероятность того, что классифицируемый поток окажется принадлежащим класса $z$,

$\hat{z}$ – прогнозируемый класс,

$P(x_i|z)$ – вероятность того, что поток класса $z$ имеет признак $x_i$.

При использовании *GNB* распределение признаков предполагается гауссовым.

#### *3.1.1.6 Алгоритм Logistic Regression*

Алгоритм логистической регрессии (*LR, Logistic Regression* [107]) представляет собой линейный метод классификации. Прогнозируемый класс $\hat{z}$ представляется в виде линейной функции:

$$\hat{z} = w_0 + w_1 x_1 + \cdots + w_m x_m, \tag{3.17}$$

где $w = (w_0; w_2; \ldots w_m)$ – вектор весовых коэффициентов.

С помощью логистической функции описываются возможные вероятности для определенного потока.

Для классификации минимизируется функция затрат (3.18):

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{m} log(e^{-z_i(X_i^T + c)} + 1), \tag{3.18}$$

где $C$ –параметр регуляризации.

#### *3.1.1.7 Алгоритм kNN*

При классификации алгоритмом «*k*-ближайших соседей» (*kNN, Nearest Neighbours* [108]) каждому объекту обучающей выборки присваивают определённые координаты в зависимости от его вектора признаков. Для классификации исследуемого потока, ему также присваивают координаты по его признакам и вычисляют расстояние до каждого из объектов обучающей выборки. Далее выбирается *k* ближайших объектов – «соседей», расстояние до которых минимально. В качестве результата классификации используют наиболее распространённый класс среди «k-ближайших соседей».

В качестве основных регулируемых параметров используют число «соседей» *k*, методы подсчета расстояний в координатной плоскости, среди которых чаще всего выделяют *Евклидово* и *Манхэттенское* расстояние. Также одним из параметров считается настройка весов в зависимости от отдаленности объектов.

#### *3.1.1.8 Алгоритм Neural network: Multi-layer Perceptron*

Среди алгоритмов машинного обучения, относящихся к нейронным сетям, особое место занимается «многослойный персептрон» (*Neural network: Multi-layer Perceptron, MLP* [109; 110]). На Рисунке 3.7 показана архитектура такой сети, которая реализована с помощью входного слоя, выходного и нескольких промежуточных скрытых слоев. Входной слой состоит из набора нейронов на основе признаков потока (вектор *A*). Каждый нейрон последующих скрытых слоев использует взвешенное линейное суммирование $w_0 + w_1 x_1 + \cdots + w_m x_m$ с нелинейной функцией активации ($g(x)$). Результат классификации получается на выходном слое.

*MLP* имеет много общего с логистической регрессией, но в отличии от нее, у *MLP* существует один или более скрытых нелинейных слоев.

В качестве основных параметров регуляризации персептрона выступает количество скрытых слоев.

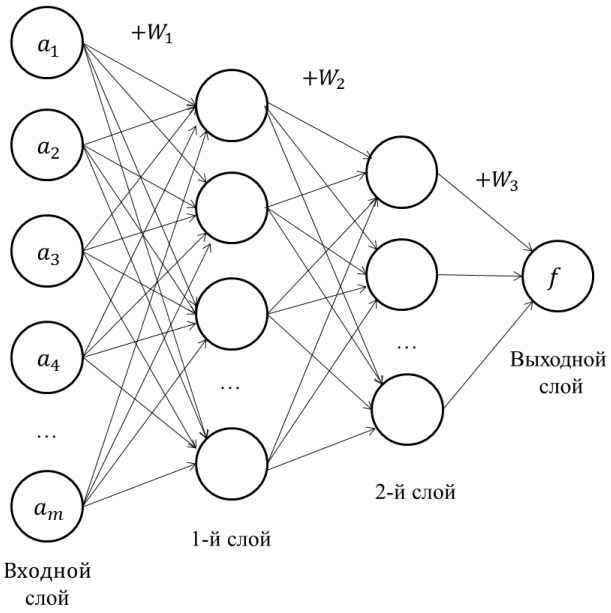
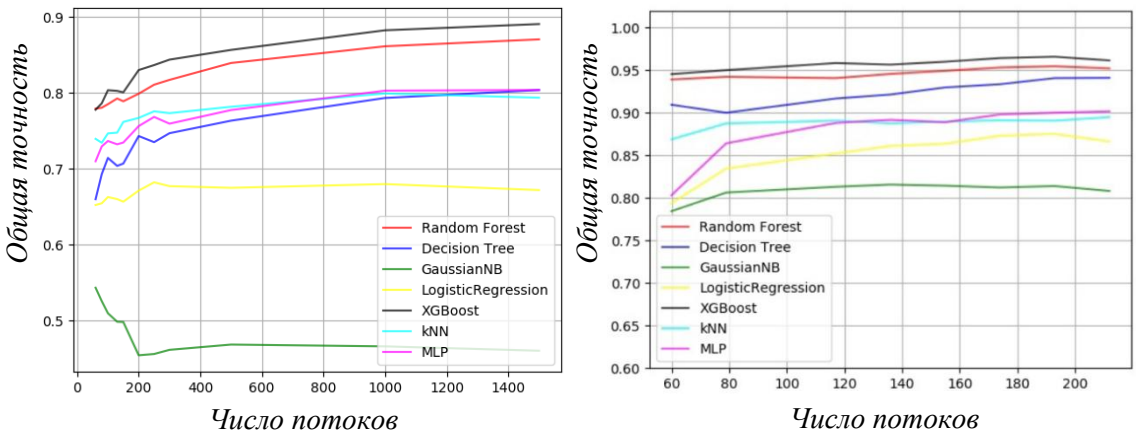



Рисунок 3.7 - Многослойный персептрон (MLP)

### 3.1.2. Результаты классификации трафика различными методами машинного обучения

Одним из важных факторов для построения модели классификации трафика является размер обучающей выборки. На Рисунке 3.8 изображена зависимость ***точности*** классификации трафика *TCP* и *UDP*-приложений от числа потоков обучающей выборки.




а) б)

Рисунок 3.8 - Зависимость ***точности*** классификации различными алгоритмами ML от количества потоков для *TCP* (а) и *UDP*-приложений (б)

Для *TCP*-потоков ***точность*** классификации с увеличением числа потоков повышается для всех алгоритмов, кроме Наивного Баейса. Для него наилучшие результаты достигаются при 60 потоках, а далее ***точность*** падает. Эта ситуация характерна для такого типа классификатора, т.к. ему не требуется большого объема информации для построения модели. Для *UDP* ***точность*** классификации при 200 потоках еще не стабилизируется и похожа на ***точность*** классификации для *TCP*-приложений для 400 потоков. Но, несмотря на это, ***точность*** остается на достаточно высоком для классификации уровне, что с одной стороны объясняется особенностями приложений, передающихся поверх *UDP*-протокола, а с другой – количеством анализируемых приложений – 8 (для *TCP* - 13).

На Рисунке 3.9 представлены результаты ***точности*** классификации для *TCP* и *UDP* наборах данных, полученные после регуляризации всех параметров и прохождения процедуры предварительной обработки данных. В Таблице 3.1 приведены основные характеристики алгоритмов, при которых удалось достичь таких результатов.

Подробные результаты и карта рекомендаций по применению регулирующих процедур для алгоритмов *ML* были получены совместно с Маньковым В.А. и представлены в статье [111]. Так как наиболее эффективными в условиях решаемой задачи оказались методы RF и XGBoost, то дальнейший анализ статической модели классификатора приводится непосредственно для них.

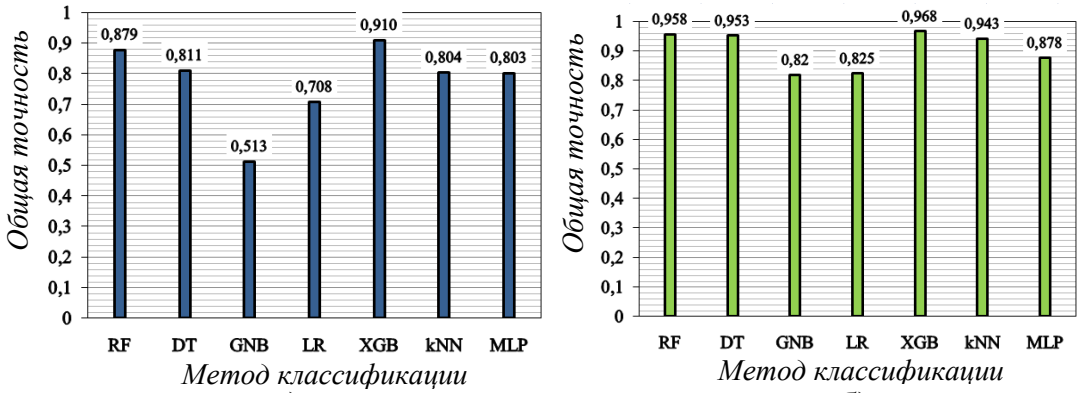



Рисунок 3.9 - Наилучшие показатели ***общей точности*** при разных методах машинного обучения для *TCP* (а) и *UDP*-приложений (б)

Таблица 3.1. Основные характеристики используемых моделей классификации трафика

| Название алгоритма *ML* | Дополнительные настроенные характеристики |
|---|---|
| *«Случайный лес» (Random Forest)* | *n_estimators=190 (дляUDP), 175 (дляTCP); max_depth=8 (дляUDP), 19 (дляTCP); min_samples_split=6;min_samples_leaf=1* |
| *«Решающее дерево» (Decision Tree)* | *n_estimators=100; max_depth=24* |
| *Гауссовский Наивный Байес (Gaussian NB)* | - |
| *Логистическая регрессия (Logistic Regression)* | - |
| *«Экстремальный градиентный бустинг» (XGBoost, XGB)* | *n_estimators=85 (дляUDP), 155 (дляTCP); max_depth=7(дляUDP), 10 (дляTCP); colsample_bytree=0,4(дляUDP), 0,65 (дляTCP); min_child_weight=2(дляUDP), 1 (дляTCP); learning_rate=0,2; subsample=0,65* |
| *Метод k-ближайших соседей (kNN)* | *k=3*; манхэттенское расстояние; веса устанавливаются в зависимости от удаленности соседей: ближайшие объекты имеют большее влияния |
| *Многослойный персептрон (MLP)* | Количество скрытых слоев – 100 |

## 3.2. Блок предварительной обработки данных

Многие методы *Machine Learning* оказываются чувствительными к выбросам или к большим разбросам данных, поэтому, часто применяются различные методы предварительной обработки трафика. Существует некоторое

количество работ, дающих представление о предварительной обработки данных сетевого трафика, но в основном эти работы посвящены классификации трафика в целях сетевой безопасности.

В [112] предварительная обработка данных определяется как алгоритм, состоящий из нескольких этапов: обработка пропущенных значений, кодирование номеров портов и протокола, определение коррелирующих признаков и обнаружение выбросов. Но в работе не приводятся результаты применения различных методов или сравнение и анализ каких-либо других подходов для предварительной обработки данных, результатом работы является подготовленный для классификации набор данных.

В [113] в качестве основного инструмента предобработки данных выступает самоорганизующаяся карта Кохонена для целей предварительной разделения на классы, и методы обнаружения и удаления выбросов. В [114] рассматривается нормализация по четырем схемам: *Frequency, Maximize, Mean Range и Rational* методами *Naïve Bayes* и *J.48* (разновидность *Decision Tree*). В [115] лучшим из походов оказалась стандартизация данных для классификаторов, построенных на основе *kNN, PCA* (метод главных компонент или *Principal Component Analysis* [116]) и *SVM* (метод опорных векторов или *Support Vector Machine* [117]), но предварительная обработка данных применяется исключительно для одного из признаков, а не для всех.

Как показано в Разделе 1, классификация трафика с целью определения *QoS* и с целью обнаружения вторжений имеют значительную разницу во многих деталях, поэтому требуются более подробный анализ.

### 3.2.1. Методы предварительной обработки данных

Среди методов предварительной обработки трафика можно выделить следующие основные направления:

1. Масштабирование – сложение или вычитание некоторой константы, затем умножение или деление на другую константу, с целью преобразования данных в единый масштаб. Это позволяет избежать больших разбросов в данных.

2. Нормализация данных – деление на норму вектора. Часто нормализацию приравнивают к масштабированию по минимуму таким образом, чтобы все значения лежали между 0 и 1.

3. Стандартизация – приведение данных к «стандартному» виду, со средним значением 0 и стандартным отклонением 1.

4. Работа с выбросами.

В работе исследуются представленные ниже методы предварительной обработки данных, разработанные на основе этих направлений и методов встроенной библиотеки *Sklearn* языка *Python* [118].

#### *3.2.1.1. Удаление выбросов Out*

Пусть матрица признаков обозначена как $A$ (3.19):

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \cdots & \cdots & a_{ij} & \cdots \\ a_{k1} & a_{k2} & \cdots & a_{km} \end{bmatrix}, \tag{3.19}$$

где нумерация $j \in [1; m]$ – признаки, а $i \in [1; k]$ – потоки трафика.

Тогда $\forall i \in [1; k]$ медиана $M_i$, а стандартное отклонение рассчитывается как (3.20):

$$\sigma_i = \sqrt{\frac{m \cdot \sum_{j=1}^{m} a_{ij}^2 - \left(\sum_{j=1}^{m} a_{ij}\right)^2}{m(m-1)}}. \tag{3.20}$$

Метод ***Out*** подразумевает замену значений, отличающихся от стандартного отклонения $\sigma_i$, на медиану $M_i$ (3.21):

$$a_{ij}^{out} = \begin{cases} a_{ij}, & \forall \ a_{ij} \leq a_{ij} \pm \sigma_i \\ M_i, & \forall \ a_{ij} \geq a_{ij} \pm \sigma_i \end{cases}. \quad (3.21)$$

#### *3.2.1.2. Удаление выбросов Out, 3 STD*

***Удаление выбросов Out, 3 STD*** предполагает замену значений, отличающихся от тройного стандартного отклонения $3\sigma_i$, на медиану $M_i$ (3.22):

$$a_{ij}^{3out} = \begin{cases} a_{ij}, & \forall \ a_{ij} \leq a_{ij} \pm 3\sigma_i \\ M_i, & \forall \ a_{ij} \geq a_{ij} \pm 3\sigma_i \end{cases}. \quad (3.22)$$

#### *3.2.1.3. Normalizer*

***Normalizer*** подразумевает нормализацию по каждой строке отдельно (3.23):

$$a_{ij}^{normal} = \frac{a_{ij} - M_i}{\sigma_i}. \quad (3.23)$$

#### *3.2.1.4. Масштабирование Standart*

Пусть $M$- медиана матрицы $A$ (3.19) и стандартное отклонение рассчитывается по всей матрице (3.24):

$$\sigma = \sqrt{\frac{m \cdot k \cdot \sum_{i=1}^{k} \sum_{j=1}^{m} a_{ij}^2 - \left(\sum_{i=1}^{k} \sum_{j=1}^{m} a_{ij}\right)^2}{m \cdot k \cdot (m-1) \cdot (k-1)}}. \quad (3.24)$$

Тогда данные после масштабирования ***Standart*** рассчитываются по формуле (3.25), так, что данные приводятся к стандартному виду со средним значением 0 и стандартным отклонением 1 $\forall \ i \in [1; k], \mathrm{j} \in [1; m]$:

$$a_{ij}^{standart} = \frac{a_{ij} - M}{\sigma}. \quad (3.25)$$

#### *3.2.1.5. Масштабирование MinMax*

Пусть минимальное (3.26) и максимальное (3.27) значение для матрицы $A$ (3.19) обозначается как:

$$a_{min} = \min_{i\in[1;k],j\in[1;m]} a_{ij}\,, \tag{3.26}$$

$$a_{max} = \max_{i\in[1;k],j\in[1;m]} a_{ij}\,. \tag{3.27}$$

Далее данные масштабируются таким образом (3.28), чтобы они находились в диапазоне [0;1]:

$$a_{ij}^{MinMax} = \frac{a_{ij} - a_{min}}{a_{max} - a_{min}}. \tag{3.28}$$

#### *3.2.1.6. Масштабирование Robust*

Масштабирование ***Robust*** – вид масштабирования данных относительно 25-го и 75-го процентиля:

$$a_{ij}^{Robust} = \frac{a_{ij} - Q_1(A)}{Q_3(A) - Q_1(A)}, \tag{3.29}$$

где $Q_1(A)$ – 1-й квартиль или 25-й процентиль по матрице $A$ (3.19), $Q_3(A)$ – 3-й квартиль или 75-й процентиль, а $IQR = Q_3(A) - Q_1(A)$-интерквартильный размах.

#### *3.2.1.7. Power – трансформация*

***Power*** – трансформация основана на отображении данных на нормальное распределение с помощью степенных преобразований. В диссертации используется преобразование Йео-Джонсона, позволяющее работать с любыми действительными числами, в т.ч. с нулевыми и отрицательными значениями [119].

Пусть $a_{ij} \in A$ (3.19), а $a_{ij}^{power}$ – отображение $a_{ij}$ на нормальное распределение, рассчитываемое исходя из соотношений (3.30):

$$a_{ij}^{power} = \begin{cases} \dfrac{\left[(a_{ij}+1)^{\lambda}-1\right]}{\lambda}, & \text{при } \lambda \neq 0, \quad a_{ij} \geq 0, \\ \ln(a_{ij}+1), & \text{при } \lambda = 0, \quad a_{ij} \geq 0, \\ -\dfrac{\left[(-a_{ij}+1)^{2-\lambda}-1\right]}{2-\lambda}, & \text{при } \lambda \neq 2, \quad a_{ij} < 0, \\ -\ln(-a_{ij}+1), & \text{при } \lambda = 2, \quad a_{ij} < 0. \end{cases} \tag{3.30}$$

где $\lambda$ - параметр распределения, который определяется путем оценки максимального правдоподобия так, чтобы стабилизировалась дисперсия и минимизировалась асимметрия. Логарифмическая функция праводоподобия имеет вид (3.31), при $n = k \cdot m$ – число элементов матрицы $A$:

$$l_n(\theta|a_{ij}) = -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{k}\sum_{j=1}^{m}\left(a_{ij}^{power} - \mu\right)^2 + \\ +(\lambda-1)\sum_{i=1}^{k}\sum_{j=1}^{m} sgn(a_{ij}) \cdot \ln(|a_{ij}|+1), \tag{3.31}$$

где $\hat{\theta} = \left(\hat{\lambda}, \hat{\mu}(\hat{\lambda}), \widehat{\sigma^2}(\hat{\lambda})\right)'$, а $\hat{\mu}(\hat{\lambda})$ - оценка математического ожидания (3.32) и $\widehat{\sigma^2}(\hat{\lambda})$ - дисперсии (3.33):

$$\hat{\mu}(\hat{\lambda}) = \frac{1}{m} \cdot \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{m} a_{ij}, \tag{3.32}$$

$$\hat{\sigma}^2(\hat{\lambda}) = \frac{1}{m} \cdot \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{m}\left(a_{ij}^{power} - \hat{\mu}(\hat{\lambda})\right)^2. \tag{3.33}$$

#### *3.2.1.8. Quantile – трансформация*

***Quantile*** – трансформация проводится в несколько этапов:

а) данные $a_{ij}$ матрицы $A$ (3.19) упорядочиваются от наименьшего к наибольшему значению;

б) определяется кумулятивная функция распределения $F(a_{ij})$, равномерно распределенная на отрезке [0;1];

в) высчитывается функция квантиля $G^{-1}(F(a_{ij}))$ для нормального распределения функции $G$.

При использовании данного метода распространяются наиболее частые значения, что позволяет уменьшить влияние выбросов. Т.к. преобразование нелинейное, то взаимосвязи между линейными корреляциями между переменными могут быть искажены, но данные, измеренные в разных масштабах более сопоставимы.

### 3.2.2. Исследование влияния методов предварительной обработки данных на результаты классификации трафика

Для экспериментальных исследований использовался набор данных *TCP* и *UDP*-приложений, подробно описанный в Разделе 2, а в качестве программного обеспечения выступали библиотеки *sklearn* и *xgboost* [105; 118], из методов машинного обучения, как уже было написано ранее, были выбраны *XGBoost* и *Random Forest* как наиболее перспективные алгоритмы.

Результаты ***общей точности*** при различных методах предварительной обработки указаны в Таблице 3.1, прирост ***общей точности*** для *TCP*-приложений на Рисунке 3.10, а для *UDP* - на Рисунке 3.11. По три лучших показателя выделены цветом. Результаты ***точности, полноты и F1-меры*** для всех классов представлены в Приложении Б.

Таблица 3.1. ***Общая точность*** классификации алгоритмами *Random Forest* и *XGBoost* с использованием разных методов предварительной обработки

| Методы предобработки данных | *TCP* | | *UDP* | |
|---|---|---|---|---|
| | *RF* | *XGB* | *RF* | *XGB* |
| *base data* | 0,829 | 0,842 | 0,890 | 0,955 |
| *Out* | 0,872 | 0,901 | 0,955 | 0,968 |
| *out, 3 STD* | 0,841 | 0,859 | 0,915 | 0,958 |
| *Standart* | 0,832 | 0,845 | 0,895 | 0,960 |
| *standart+out* | 0,871 | 0,896 | 0,953 | 0,965 |
| *Minmax* | 0,829 | 0,844 | 0,898 | 0,958 |
| *minmax+out* | 0,875 | 0,893 | 0,953 | 0,963 |
| *Robust* | 0,830 | 0,842 | 0,890 | 0,955 |
| *robust+out* | 0,872 | 0,897 | 0,953 | 0,963 |
| *Power* | 0,832 | 0,844 | 0,918 | 0,950 |
| *power+out* | 0,872 | 0,910 | 0,953 | 0,968 |
| *Quantile* | 0,830 | 0,848 | 0,895 | 0,955 |
| *quantile+out* | 0,879 | 0,909 | 0,958 | 0,968 |
| *Normalizer* | 0,795 | 0,803 | 0,908 | 0,913 |
| *normalizer+out* | 0,845 | 0,863 | 0,940 | 0,938 |

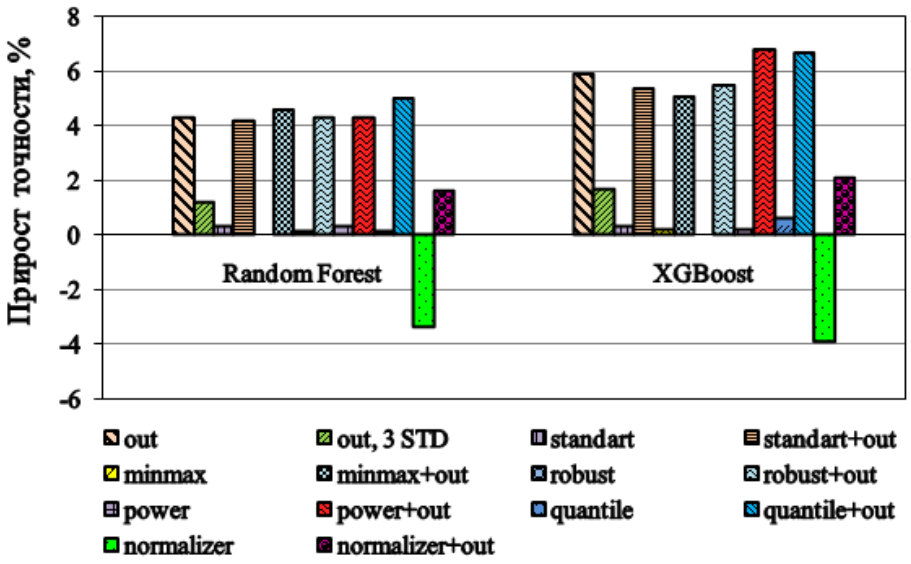



Рисунок 3.10 – Прирост ***точности*** классификации *TCP*-потоков алгоритмами *RF* и *XGB* с использованием разных методов предварительной обработки

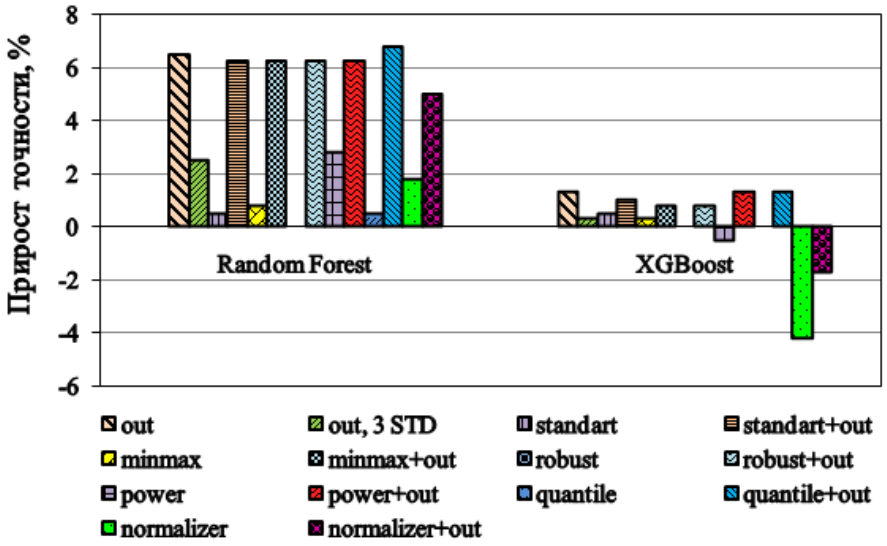



Рисунок 3.11 – Прирост ***точности*** классификации *UDP*-потоков алгоритмами *RF* и *XGB* с использованием разных методов предварительной обработки

Здесь ***base data*** – означает результаты классификации без предварительной обработки данных. Диаграммы и таблицы показывают повышение эффективности классификации при обоих методах работы с выбросами (и ***out***, и ***out 3 STD***), но лучшие показатели были достигнуты при методе ***out***, т.е. когда в качестве выброса считается значение, превышающее СКО. Например, для *Random Forest* ***точность*** классификации *TCP*-приложений с помощью out увеличивается на 4,1%, а *XGBoost* – на 5,9 %, а при ***out, 3STD*** на 1,2% и 1,7%, что уже не так существенно. Поэтому для дальнейших исследований в качестве методов работы с выбросами – метод ***out***. Остальные процедуры анализировались с участием метода ***out*** и без. Как видно из результатов, при использовании метода ***out*** все процедуры показывают ***точность*** выше, чем без. Например, для ***Normalizer*** ***точность*** увеличивается на 5% для *RF* и 6% для *XGBoost*. Наибольшее значение удаление выбросов оказывает на приложения *Skype* и *Telnet* – повышение эффективности результатов достигает 33,8%.

Стандартизация с помощью ***Normalizer*** показывает наихудшие показатели по сравнению с остальными методами предварительной обработки данных. С учетом, что метод ***Standart*** работает сравнимо с просто методом ***out*** и не вносит ухудшений, то вероятная причина понижения точности классификации для

***Normalizer*** заключается в подходе к нормализации по отдельным признакам. Таким образом, моменты случайной величины рекомендовано рассчитывать для всей матрицы целиком, а не по поточно.

Из методов масштабирования при классификации ***TCP***-приложений методом *RF* наиболее эффективным оказалась процедура ***MinMax***, а для *XGB* – ***Robust***, используемые совместно с процедурой ***out***.

Оба метода трансформации: ***Power*** и ***Quantile*** показали схожие результаты, но процедура ***Quantile*** оказалась более эффективной. ***Точность Quantile+out*** по сравнению с методом ***out*** без других алгоритмов выше на 0,7 и на 0,9 % для *RF* и ***XGB*** соответственно. Такой результат кажется незначительным, но если проанализировать ***F1***-меру по отдельным классам (Приложение Б), можно заметить, что показатели *SSL* повышаются на 4-5%, а *RTMP* на 2,4-2,7%, а общая точность при этом падает из-за незначительных снижений на 0,5-1% в ***F1***-мере для *Apple, HTTP_Proxy* и *IMAPS*. Поэтому важно использовать не только ***out***, но и ***Quantile***-трансформацию.

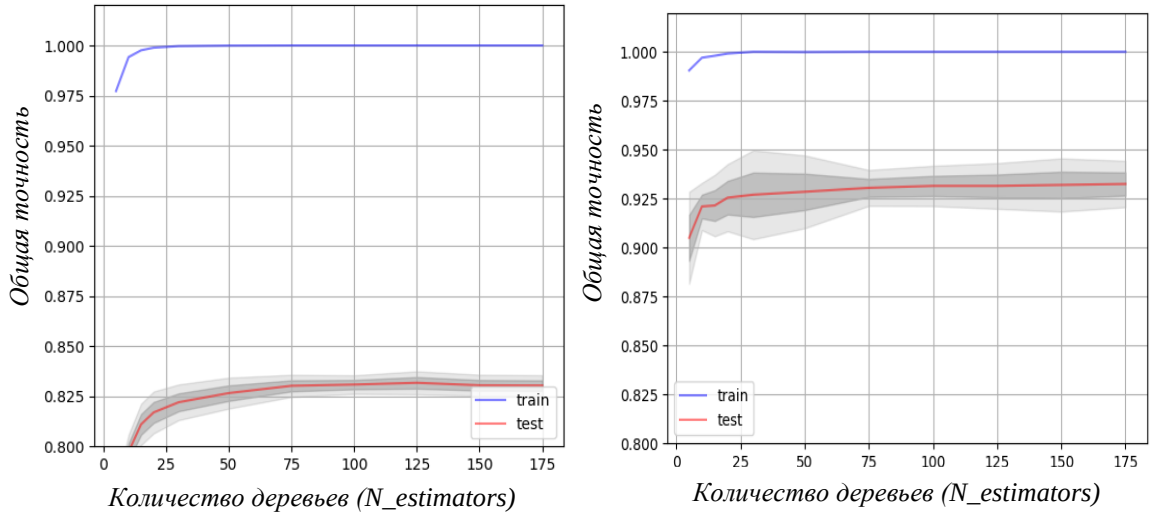
Несмотря на то, что методы *RF* и *XGB* обычно довольно устойчивые к выбросам и различиям в масштабах, при классификации сетевого трафика рекомендуется выполнять предварительную обработку данных, т.к. она позволяет повысить эффективность классификации. В частности, предлагается использовать метод ***Quantile***–трансформации совместно с ***out*** – работой с выбросами, повышая ***точность*** классификации *TCP*–потоков на 6,8% для *XGB* и на 5% для *RF*, а *UDP* на 1,3% и 6,8% соответственно.

## 3.3. Блок классификации

### 3.3.1. Настройка алгоритма Random Forest

Для исследования влияния параметров математической модели на результат классификации были выбраны *TCP* и *UDP*–потоки, подробно описанные в Подразделе 2.3.

На Рисунке 3.12 изображена зависимость ***точности*** по методу *OOB* для тестовой и обучающей выборки в зависимости от количества деревьев (*N_estimators*), а на Рисунке 3.13 – зависимость отклонения от средней ошибки *OOB* от количества деревьев, максимальному количеству просмотренных в ветвях признаках (*max_features*) и критерию примеси (индекс Джини или Энтропия). Темно-серым отмечена область, в которой находятся значения в пределах стандартного отклонения, а светло–серым - в пределах максимального. В качестве возможных вариантов *max_features* рассматриваются стандартные варианты: *max_features*=$\sqrt{m}$, *max_features*=$log_2 m$ и *max_features*=*m*.

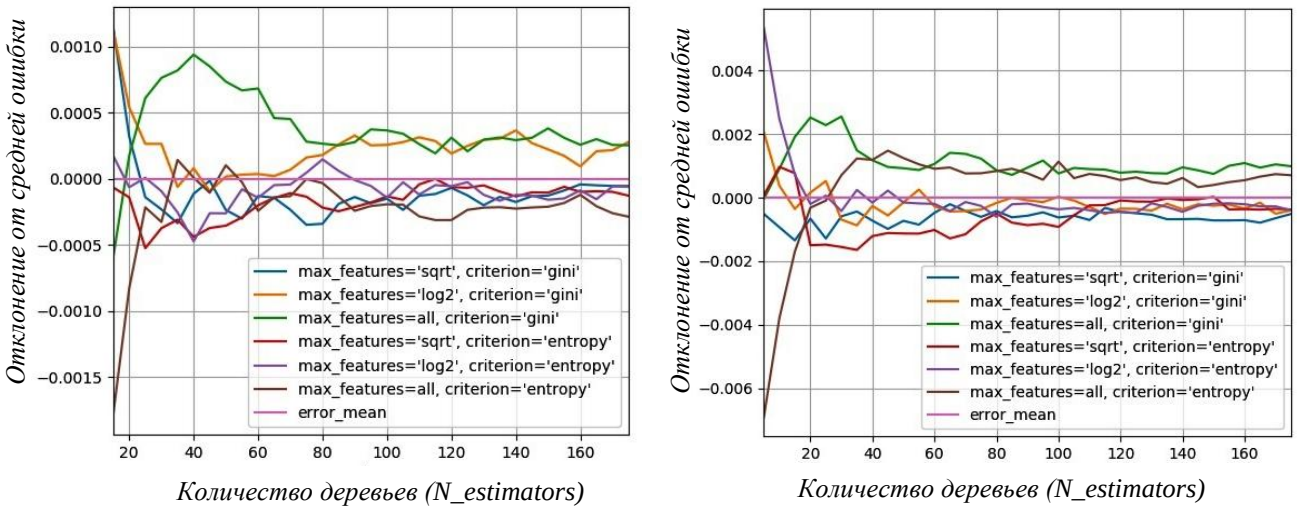



а) б)

Рисунок 3.12 - Зависимость ***общей точности*** классификации от количества деревьев (*N_estimators*) для *TCP* (а) и *UDP*-потоков (б)

Как видно из Рисунка 3.12, для *TCP*-потоков наивысшая ***точность*** 0,83 достигается при 125 деревьях и выше, а для *UDP*–0,93 при 175 деревьях и выше. При дальнейшем росте числа деревьев ***точность*** практически не меняется, но значительно увеличивается сложность расчетов и время построения модели. Результаты тренировочного *TCP*-набора принимают значение 1, а тестового – незначительно колеблется вблизи 0,82. Это говорит о том, разброс в ошибке составляет около 18%, а модель является переобученной. В случае с *UDP* ошибка

8%, но модель также является переобученной. Для того чтобы справиться с этой проблемой и уменьшить разброс между тестовыми и тренировочными результатами, необходимо воспользоваться механизмами регуляризации параметров.

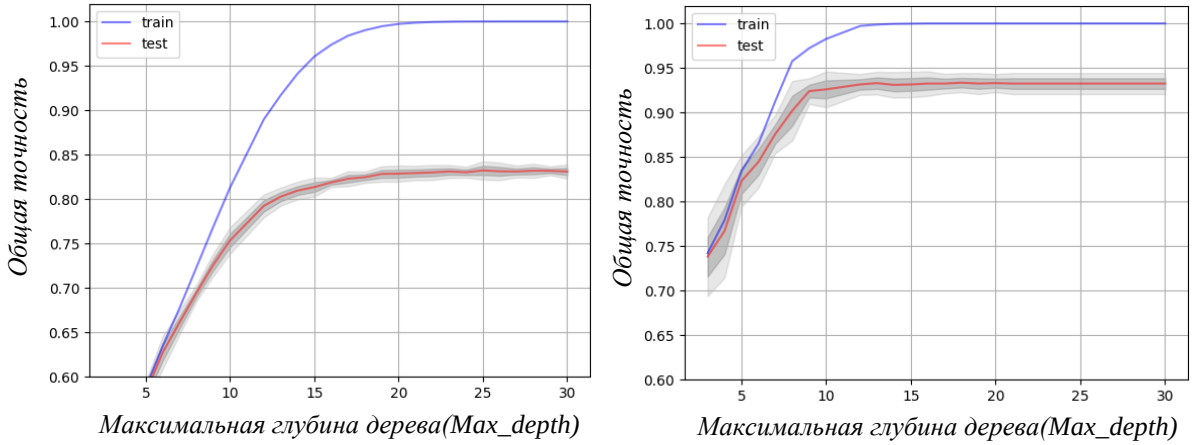



а) б)

Рисунок 3.13 - Зависимость отклонения от средней ошибки классификации от количества деревьев (*N_estimators*), максимального числа признаков для разветвления и критерия примеси (индекс Джини или Энтропия) для *TCP* (а) и *UDP*-потоков (б)

При 125 деревьях для *TCP*-потоков небольшой *OOB*-ошибки можно достичь при *max_features*=$\sqrt{m}$ с индексом Джини или *max_features*=$m$ с Энтропией. При увеличении количества деревьев ошибка незначительно снижается, но сложность линейно возрастает. Для набора данных *UDP* при 175 деревьях наименьшая *OOB*-ошибка достигается при *max_features*=$\sqrt{m}$ с Энтропией или *max_features*=$log_2 m$ с индексом Джини.

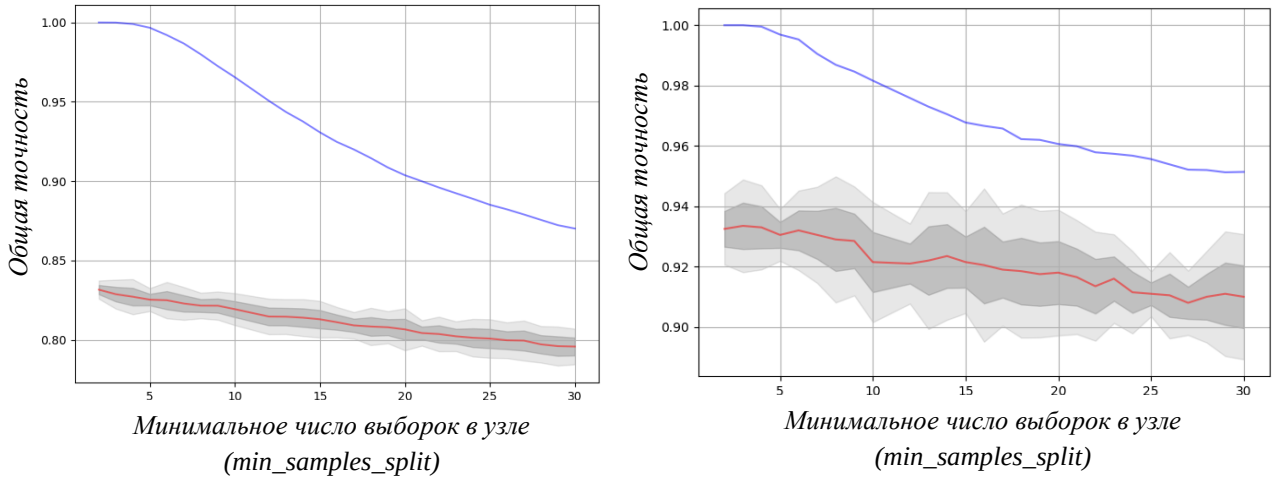
На Рисунке 3.14 изображены валидационные кривые для *TCP* и *UDP*-приложений в зависимости от глубины деревьев. С увеличением глубины деревьев ***точность*** классификации возрастает. Для *TCP*-потоков *max_depth* начинается с *max_depth=25*, а для *UDP*-потоков с 17.




а) б)

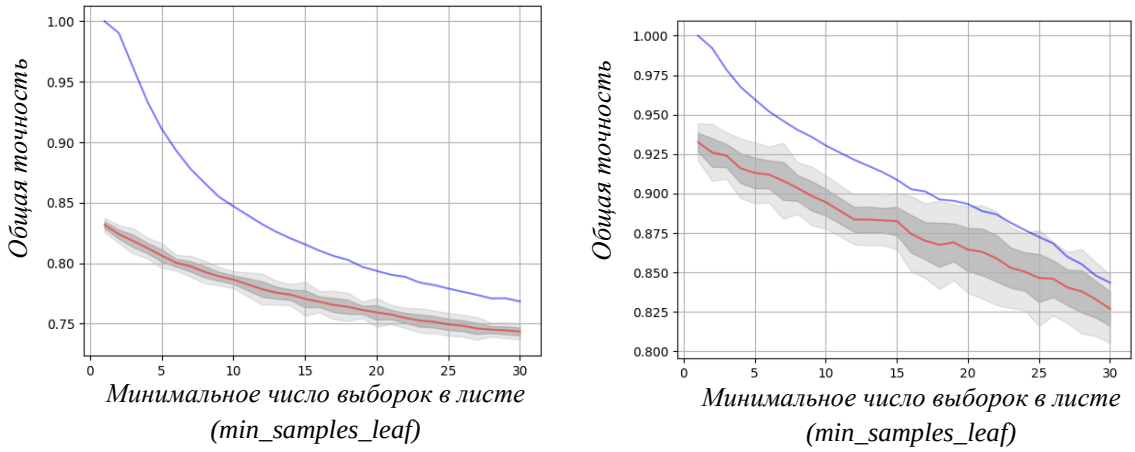
Рисунок 3.14 - Зависимость ***общей точности*** классификации от максимальной глубины деревьев (*Max_depth*) для *TCP* (а) и *UDP*-потоков (б)

На Рисунке 3.15 показаны кривые ***точности*** для минимального числа выборок в узле, а на Рисунке 3.16 – для минимального числа выборок в листе. При увеличении этих параметров ***точность*** классификации незначительно уменьшается. Для *TCP*-потоков *min_samples_split=2*, а для *UDP* - 4. Для обоих наборов данных *min_samples_leaf=1*.




а) б)

Рисунок 3.15 - Зависимость точности классификации от минимального числа выборок в узле (*min_samples_split*) для *TCP* (а) и *UDP*-потоков (б)




а) б)

Рисунок 3.16 - Зависимость точности классификации от минимального количества выборок в листе (*min_samples_leaf*) для *TCP* (а) и *UDP*-потоков (б)

Но, несмотря на уменьшение ***точности***, разброс между результатами тестовой и обучающей выборки тоже уменьшается и модель уже не такая переобученная. Поэтому для набора данных *TCP*-приложений предлагается использовать *max_depth=19, min_samples_split=7, min_samples_leaf=1*, сохраняя при этом ***точность*** 0,82, но уменьшая переобучение на 18%. Немного увеличив количество деревьев и количество максимальных признаков до *max_features=13, n_estimators=175,* ***точность*** станет равной 0,87, т.е. увеличится на 5%.

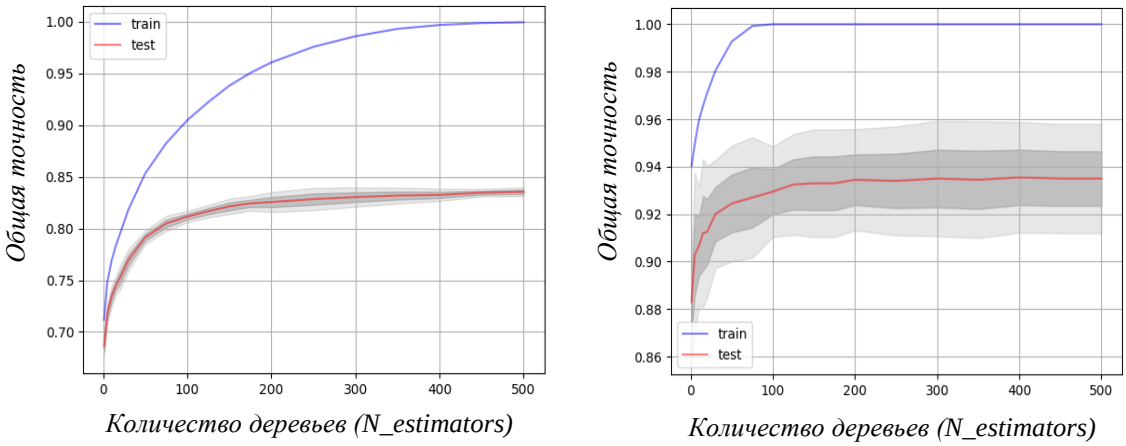
Аналогичным образом для набора данных *UDP*-потоков можно определить: *n_estimators=190, max_depth=8, min_samples_split=6 и min_samples_leaf=1*.

### 3.3.2. Настройка алгоритма XGBoost

Для реализации алгоритма *XGBoost* используется одноименная сторонняя библиотека [105].

На Рисунке 3.17 изображены валидационные кривые, показывающие ***точность*** классификации в зависимости от числа деревьев для наборов данных *TCP* и *UDP*. В отличие от алгоритма *Random Forest*, для достижения высокой точности классификации не требуется большого количества деревьев и разброс

между тестовой и обучающей выборкой не так велик. При построении деревьев с максимальной глубиной равной 5, точность 80% достигается при 50 деревьях для *TCP* и 90% при 5 деревьях для *UDP*.

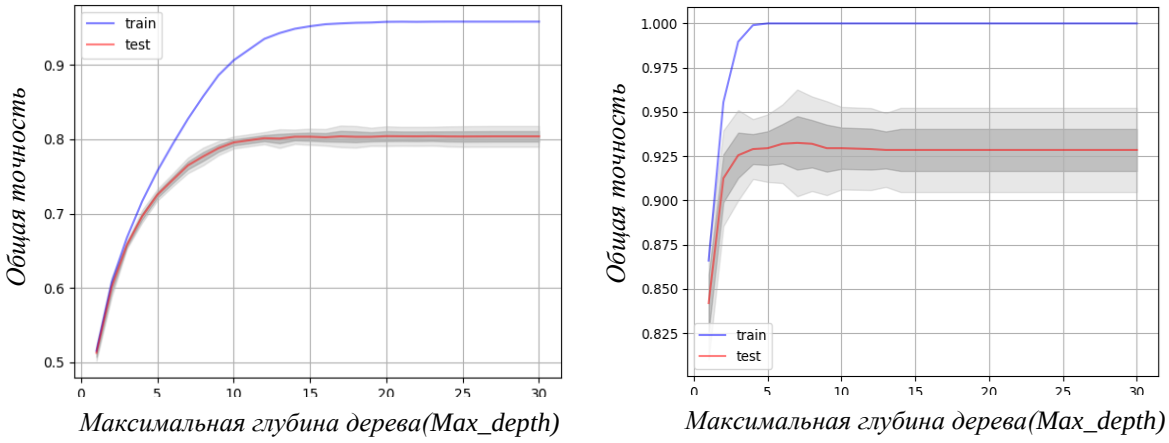



а) б)

Рисунок 3.17 - Зависимость ***точности*** классификации от количества деревьев (*N_estimators*) для *TCP* (а) и *UDP*-потоков (б)

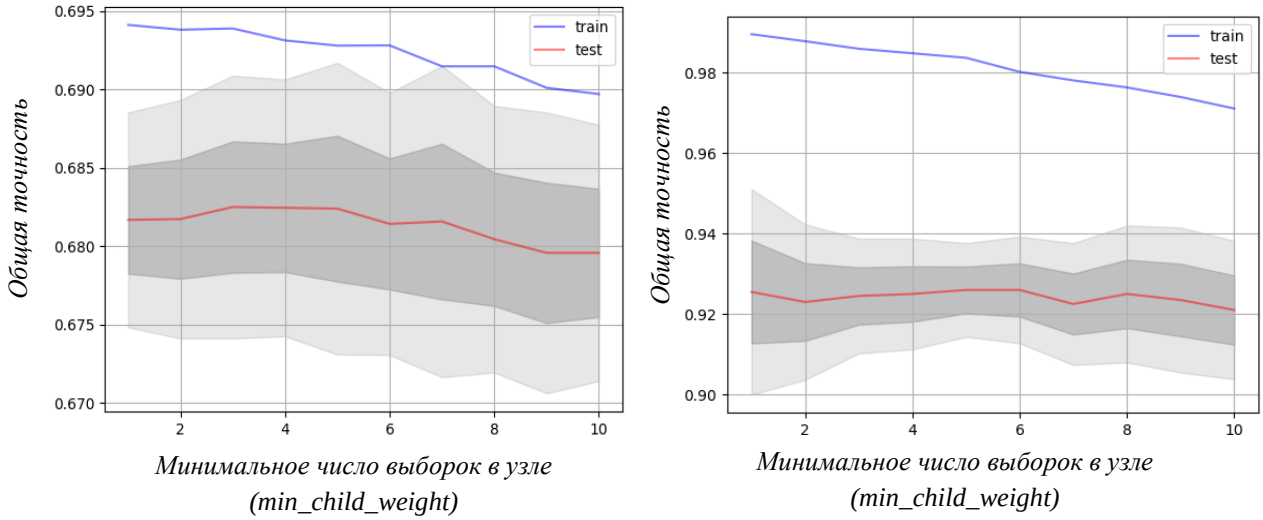
На Рисунке 3.18 изображены зависимости ***точности*** классификации от максимальной глубины деревьев. Графики для *XGBoost* очень похожи на аналогичные графики для *Random Forest* (Рисунок 3.12), но рост точности для *XGBoost* начинается при значительно меньших значения *max_depth:7-10* для *TCP* и 3-5 для *UDP*.

На Рисунке 3.19 показаны валидационные прямые для *min_child_weight* – параметра, отвечающего за минимальное число выборок в каждой вершине - аналогично *min_samples_leaf* для *Random Forest*. Также как и в случаях с *Random Forest*, этот параметр помогает справиться с переобучением.




а) б)

Рисунок 3.18 - Зависимость ***точности*** классификации от максимальной глубины деревьев (*Max_depth*) для *TCP* (а) и *UDP*-потоков (б)

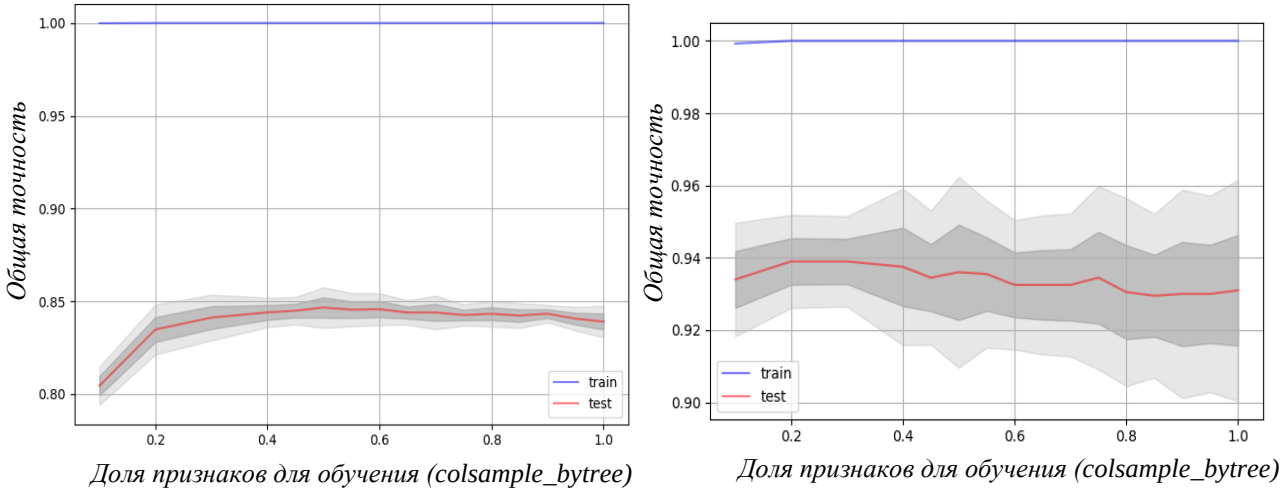



а) б)

Рисунок 3.19 - Зависимость ***точности*** классификации от минимального числа выборок в каждой вершине (*min_child_weight*) для *TCP* (а) и *UDP*-потоков (б)

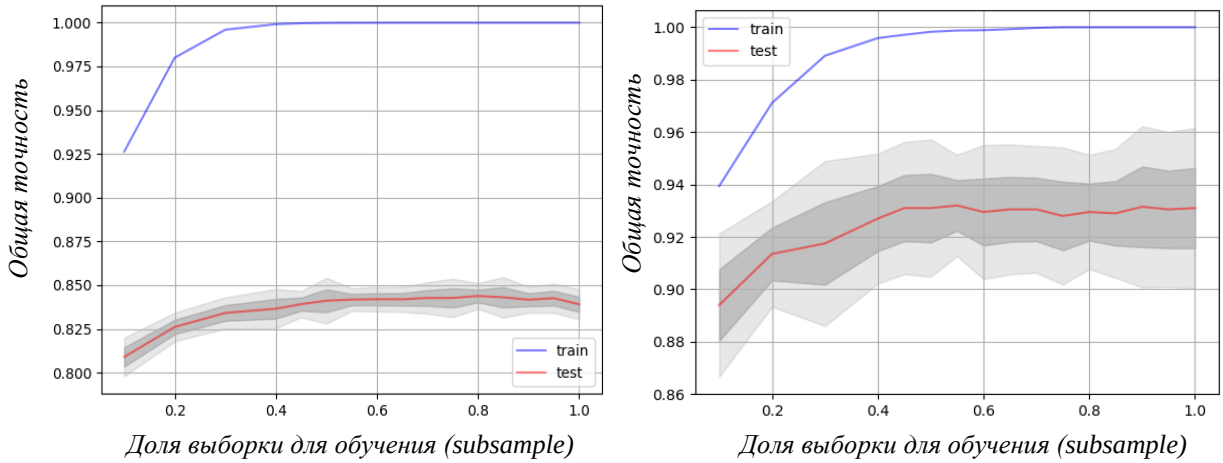
На Рисунке 3.20 изображены зависимости точности классификации трафика от доли признаков, используемых для построения деревьев (*colsample_bytree*), аналогично *max_features* для *Random Forest*, а на Рисунке 3.21 – зависимости точности классификации от доли выборки для построения деревьев (*subsample*), аналогично методу *OOB*. Параметры *colsample_bytree* и *subsample* добавляют

случайность в процесс обучения, благодаря этому, при изменении этих долей, можно добиться лучших результатов моделирования.




а) б)

Рисунок 3.20 - Зависимость ***точности*** классификации от доли признаков, используемых для обучения деревьев (*colsample_bytree*) для *TCP* (а) и *UDP* (б)




а) б)

Рисунок 3.21 - Зависимость ***точности*** классификации от доли выборки, используемой для обучения каждого дерева (*subsample*) для *TCP* (а) и *UDP* (б)

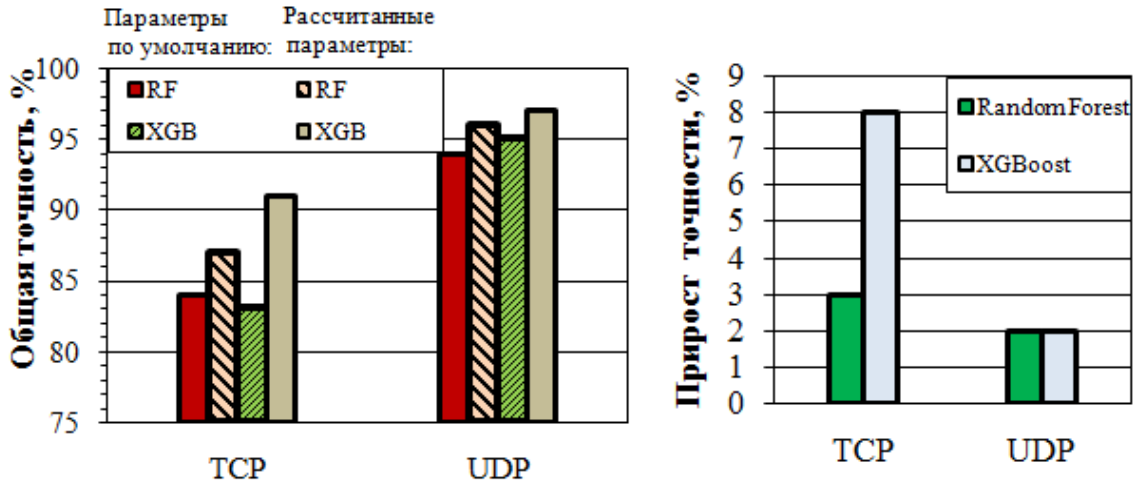
Путем подбора параметров, можно определить те, при которых достигается наилучшая ***точность***. Для *TCP*-потоков ***точность*** 0,9005 может быть получена при *n_estimators*=155, *max_depth*=10, *colsample_bytree*=0,65, *min_child_weight*=1,

*subsample*=0,65; а для *UDP*-потоков ***точность*** 0,9675 – при *n_estimators*=85, *max_depth*=7, *colsample_bytree*=0,4, *min_child_weight*=2 и *subsample*=0,65.

### 3.3.3. Анализ влияния настройки параметров модели на результаты классификации трафика

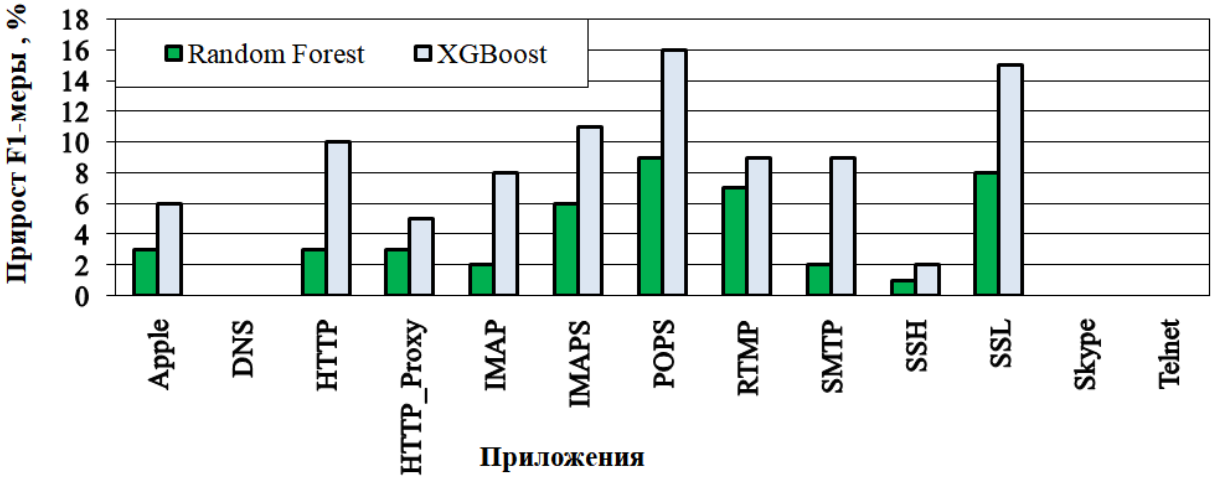
В Таблицах 3.7 и 3.8 представлены результаты классификации *TCP* и *UDP* потоков с параметрами по умолчанию и параметрами, которые были получены при изучении математической модели *Random Forest* и *XGBoost*, а на Рисунках 3.22-3.23 - их графическое представление.

Из таблиц видно, что для набора данных *TCP* улучшение результатов ***точности*** при настройке *RF* - 3,26%, а при настройке *XGBoost* – 6,69%. В случаях с набором данных *UDP*, настройка *RF* повышает ***точность*** классификации на 1,5% , а настройка *XGBoost* –на 2%. При рассмотрении результатов отдельных приложений можно заметить, что настройка *RF* повышает ***точность классов*** до 12% (для *POPS*), а *UDP* – до 11% (для *NTP*). Настройка *XGBoost* для *TCP* повышает ***полноту*** выделения ***классов*** до 19% для *POPS* и *SSL*, а для *UDP* – до 12% (для *NTP*).

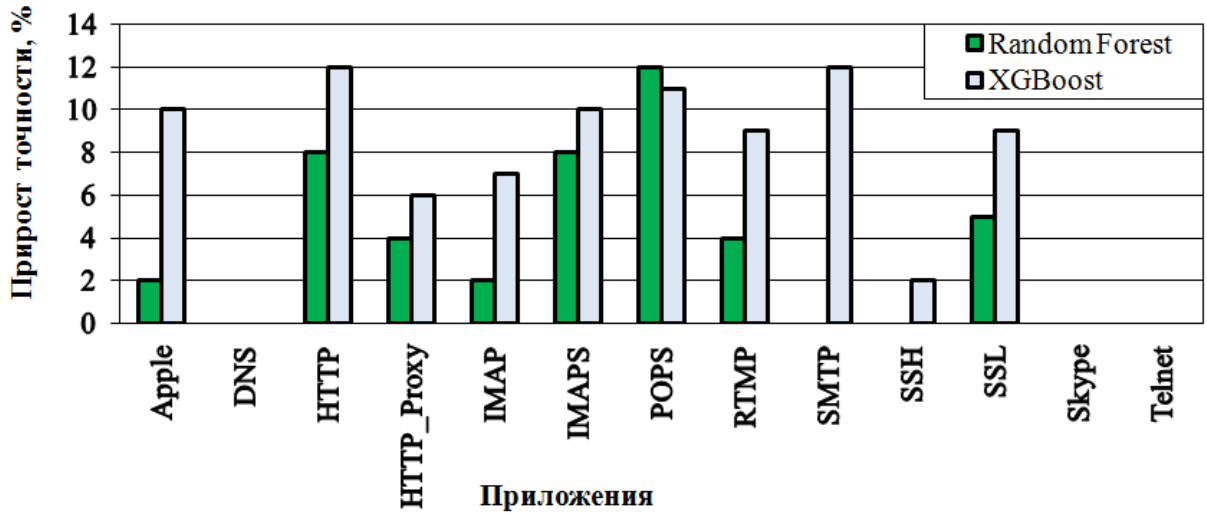



а) ***Общая точность*** б) Прирост ***точности***

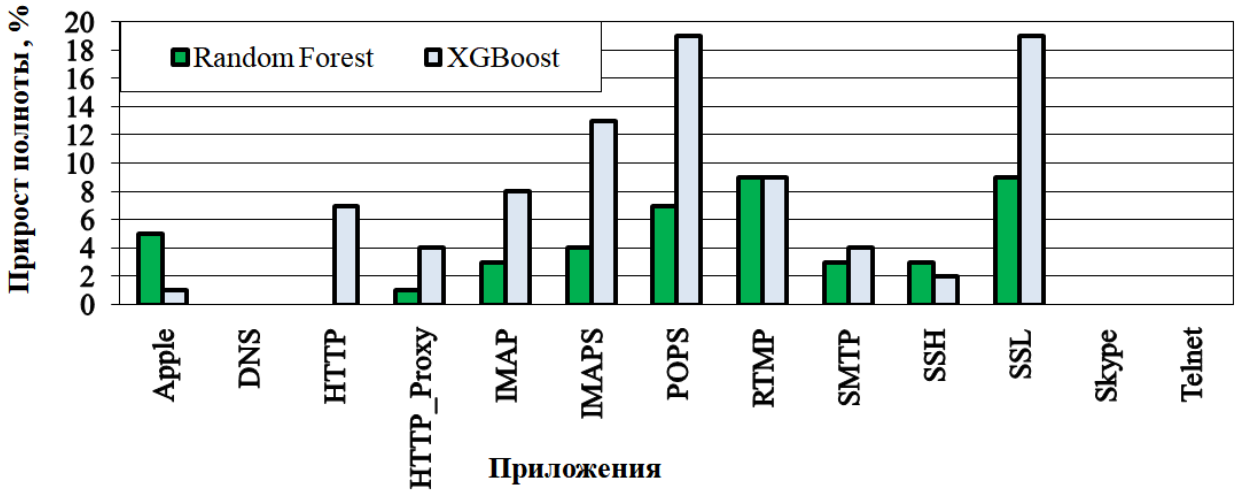
Рисунок 3.22 – Результаты классификации *TCP* и *UDP*-приложений до и после применения рассчитанных параметров




а). Прирост значения ***F1-меры***




б). Прирост ***точности классов***




в). Прирост значения ***полноты*** *выделения классов*

Рисунок 3.23 - Результаты классификации *TCP*-приложений после применения рассчитанных параметров

Таблица 3.7. Результаты классификации приложений набора данных *TCP* методами *Random Forest* и *XGBoost* с параметрами по умолчанию и с рассчитанными параметрами

| № | Приложение | *Random Forest* (параметры по умолчанию) | | | *Random Forest* (рассчитанные параметры) | | | *XGBoost* (параметры по умолчанию) | | | *XGBoost* (рассчитанные параметры) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ***Точность класса*** | ***Полнота*** | ***F1*** | ***Точность класса*** | ***Полнота*** | ***F1*** | ***Точность класса*** | ***Полнота*** | ***F1*** | ***Точность класса*** | ***Полнота*** | ***F1*** |
| 1 | *Apple* | 0,75 | 0,93 | 0,83 | 0,77 | 0,98 | 0,86 | 0,73 | 0,96 | 0,83 | 0,83 | 0,97 | 0,89 |
| 2 | *DNS* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | *HTTP* | 0,76 | 0,8 | 0,78 | 0,84 | 0,79 | 0,81 | 0,76 | 0,78 | 0,77 | 0,88 | 0,85 | 0,87 |
| 4 | *HTTP_Proxy* | 0,86 | 0,93 | 0,89 | 0,9 | 0,94 | 0,92 | 0,86 | 0,91 | 0,88 | 0,92 | 0,95 | 0,93 |
| 5 | *IMAP* | 0,86 | 0,83 | 0,85 | 0,88 | 0,86 | 0,87 | 0,83 | 0,8 | 0,81 | 0,9 | 0,88 | 0,89 |
| 6 | *IMAPS* | 0,72 | 0,74 | 0,73 | 0,8 | 0,78 | 0,79 | 0,73 | 0,71 | 0,72 | 0,83 | 0,84 | 0,83 |
| 7 | *POPS* | 0,71 | 0,58 | 0,64 | 0,83 | 0,65 | 0,73 | 0,75 | 0,53 | 0,62 | 0,86 | 0,72 | 0,78 |
| 8 | *RTMP* | 0,69 | 0,74 | 0,71 | 0,73 | 0,83 | 0,78 | 0,72 | 0,78 | 0,75 | 0,81 | 0,87 | 0,84 |
| 9 | *SMTP* | 0,89 | 0,9 | 0,89 | 0,89 | 0,93 | 0,91 | 0,81 | 0,91 | 0,85 | 0,93 | 0,95 | 0,94 |
| 10 | *SSH* | 0,97 | 0,94 | 0,96 | 0,97 | 0,97 | 0,97 | 0,95 | 0,94 | 0,94 | 0,97 | 0,96 | 0,96 |
| 11 | *SSL* | 0,69 | 0,52 | 0,59 | 0,74 | 0,61 | 0,67 | 0,69 | 0,52 | 0,6 | 0,78 | 0,71 | 0,75 |
| 12 | *Skype* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 13 | *Telnet* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | ***Общая точность*** | 0,8397 | | | 0,8723 | | | 0,8336 | | | 0,9105 | | |

Таблица 3.8. Результаты классификации приложений набора данных *UDP* методами *Random Forest* и *XGBoost* с параметрами по умолчанию и с рассчитанными параметрами

| № | Приложение | *Random Forest* (параметры по умолчанию) | | | *Random Forest* (рассчитанные параметры) | | | *XGBoost* (параметры по умолчанию) | | | *XGBoost* (рассчитанные параметры) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ***Точность класса*** | ***Полнота*** | ***F1*** | ***Точность класса*** | ***Полнота*** | ***F1*** | ***Точность класса*** | ***Полнота*** | ***F1*** | ***Точность класса*** | ***Полнота*** | ***F1*** |
| 1 | *DNS* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | *IPsec* | 0,84 | 0,76 | 0,8 | 0,95 | 0,82 | 0,88 | 0,93 | 0,76 | 0,84 | 0,92 | 0,88 | 0,9 |
| 3 | *NTP* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | *NetBIOS* | 1 | 1 | 1 | 1 | 0,98 | 0,99 | 0,98 | 1 | 0,99 | 1 | 1 | 1 |
| 5 | *Quic* | 0,82 | 0,94 | 0,88 | 0,83 | 0,96 | 0,89 | 0,83 | 0,96 | 0,89 | 0,84 | 0,98 | 0,91 |
| 6 | *SNMP* | 0,96 | 0,96 | 0,96 | 0,91 | 0,98 | 0,94 | 0,96 | 0,96 | 0,96 | 1 | 0,98 | 0,99 |
| 7 | *STUN* | 0,9 | 0,86 | 0,88 | 0,98 | 0,9 | 0,94 | 0,9 | 0,9 | 0,9 | 1 | 0,9 | 0,95 |
| 8 | *UPnP* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | ***Общая точность*** | 0,94 | | | 0,955 | | | 0,9475 | | | 0,9675 | | |

На основе этих значений можно сделать вывод, что настройка математических параметров модели является важным этапом при создании классификатора трафика. Для классификации набора данных *TCP* влияние настройки параметров оказалось большим, чем при классификации *UDP*. Это объясняется тем, что и при ненастроенных параметрах для *UDP*-потоков были показаны хорошие результаты, иногда достигающие 100%. С одной стороны, классификаторы для *UDP* учатся быстрее, а с другой стороны – для *TCP* выбрано 13 приложений против 8 для *UDP*, что усложняет задачу классификации и обучения.

В большинстве работ, связанных с классификацией трафика, наилучшие результаты перед всеми остальными алгоритмами показывает алгоритм *Random Forest* [13; 17; 120-122]. В [88] показано, что один из алгоритмов бустинга на основе деревьев (*AdaBoost*) в целом показывает результаты хуже, по сравнению с *Random Forest*. Но, судя по проведенному эксперименту заметно, что до настройки параметров модели *XGBoost* по результатам уступал даже ненастроенному *RF*, а после настройки – опередил и настроенный, и ненастроенный *RF*. Результаты работы над разделом 3.3 опубликованы в [123].

## 3.4. Сравнение статической модели классификации с другими работами

В разделе 4.6 монографии [93], посвященному методологии классификации с помощью алгоритма *Random Forest*, рекомендуется строить деревья максимальной глубины (*max_depth=max)* и использовать малое количество (около *N_estimators* ≈ 5) деревьев для небольшого состава приложений: *SSL, HTTP, DNS, BitTorrent, Steam* и *Skype*. При этом не приводятся комментарии относительно предварительной обработки данных.

В [113] для *RF* строятся деревья небольшой глубины (*max_depth* ≈ 5) на основе индекса Джини. Также упоминается, что выбросы были удалены, но определение понятия выбросов не приводится.

Статья [115] посвящена нормализации атрибутам для целей обнаружения вторжений, где статистическую нормализацию посчитали лучшим способом предварительной обработки данных. Исходя из описания, статистическая нормализация аналогична методу ***Normalizer*** (Подпункт 3.2.1.3).

Как было показано в предыдущих разделах, непосредственное сравнение результатов экспериментов из разных исследований некорректно. Поэтому создаются общие условия для классификации, а подходы к предварительной обработке данных воссоздаются по их описаниям из других работ. Важно отметить, что описанные в исследованиях процедуры применяются для классификации преимущественно в целях сетевой безопасности и имеют иную матрицу признаков, поэтому не всегда удается достичь таких же высоких результатов классификации как в оригинальных работах.

Для сравнения статической модели классификации, описанной в данной работе, с моделями других исследователей, из набора данных *TCP* (Таблица 2.2) были выбраны следующие приложения: *SSL, DNS*, *HTTP*, *Skype* и *Telnet*. Так как *XGBoost* является относительно новым алгоритм *ML*, и для него в работах не предусмотрены никакие рекомендации, то при построении моделей на его основе пользовались теми же принципами, что и для *Random Forest*. Для метода из [115] использовались параметры, установленные в классификаторе по умолчанию.

Результаты точности классификации представлены в Таблице 3.9. Видно, что для классификации с целью обеспечения *QoS* не следует строить деревья малой глубины, т.к. это значительно снижает качество классификации. В работах [93] и [115] приблизительно схожие показатели ***точности***, но модель статической классификации, предложенная в этой работе, показала результаты на 15-25% выше, что свидетельствует о верном выборе процедур предварительной обработки данных и настройки гиперпараметров применительно к матрице признаков, работающей в режиме реального времени.

Также стоит отметить, что по сравнению с Таблицей 2.7, результаты классификации поднялись с 0,805 до 0,96, т.е. на 15,5% за счет использования методов, описанных в третьем разделе.

Таблица 3.9. ***Общая точность*** классификации для *RF* и *XGB* при применении статической модели этой работы в сравнении с другими работами

| Исследования | *Random Forest* | *XGBoost* |
|---|---|---|
| эта работа | 0,960 | 0,970 |
| [93] | 0,810 | 0,820 |
| [113] | 0,758 | 0,720 |
| [115] | 0,806 | 0,793 |

Выводы по третьему разделу

1. Разработана статическая модель классификации трафика, которую предлагается использовать в режиме реального времени в сетях, в которых редко появляются новые классы. ***Точность*** классификации для небольшого количества *TCP*-приложений (около 5) - 96% (*RF*), 97% (*XGB*), для более широкого спектра *TCP*-классов (около 13) – 87,2% (*RF*), 91,1% (*XGB*), а *UDP*-классов – 95,5% (*RF*), 96,8% (*XGB*).

2. Добавлен блок предварительной обработки данных в модель классификации трафика, включающий в себя комплексный подход с использованием нормализации, стандартизации, трансформации и работы с выбросами. Для методов *Random Forest* и *XGBoost* предлагается использовать метод ***Quantile*** – трансформации совместно с ***out*** – работой с выбросами, повышая ***точность*** классификации *TCP*-приложений на 6,8% для *XGB* и на 5% для *RF*, а *UDP*-приложений на 1,3% и 6,8% соответственно.

Также для *XGBoost* возможен подход с использованием ***Power***–трансформации совместно с ***out***, ***точность*** при этом повышается на 6% для *TCP* и на 1,3% для *UDP* соответственно. Для *Random Forest* возможен подход с ***MinMax***–масштабированием, повышающий ***точность*** на 4,8% (*TCP*) и на 6,5% (*UDP*).

3. Приложения *Skype* и *Telnet* наиболее чувствительны к выбросам, поэтому требуют обязательной работы над выбросами. ***Точность*** классификации для этих приложений повышается на 26-34%.

4. Выбросами рекомендуется считать значения, превышающие среднее значение больше, чем на СКО, а не значения, превышающее среднее значение больше, чем на 3СКО.

5. Нормализация по всей матрице признаков показывает более эффективные результаты по сравнению с нормализацией по потокам.

6. Блок настройки гиперпараметров (глубина дерева, количество деревьев, минимальное количество классов в узле для разветвления и т.д.) повышает ***точность*** классификации *TCP*–потоков до 19% для метода *XGBoost*. При применении новых рекомендаций результаты работы *XGBoost* оказались выше результатов метода *Random Forest* на 3 и 1% для *TCP* и *UDP*-потоков соответственно, что означает его конкурентоспособность и возможность применения для целей классификации трафика. В работах других авторов по классификации потоков трафика наилучшие результаты модели фиксировались у метода *Random Forest*, в то время как методы «Градиентного бустинга» были на 10-40% ниже.

7. Разработанные блоки предварительной обработки данных и настройки гиперпараметров повысили ***точность*** классификации на 15,5% для малого количества приложений, что на 15-25% выше по сравнению с методами, предложенными в других работах.

Результаты работы над вторым разделом были представлены на конференции *«The Fourth International Conference of Artificial Intelligence, Medical Engineering, Education (AIMEE2020)»* и опубликованы в [111; 123].

# Раздел 4. Динамическая модель классификации трафика

В четвертом разделе ведется работа над построением модели кластеризации трафика, которая при объединении со статической моделью классификации трафика образует итоговую динамическую модель классификации трафика, показывающую высокоточные результаты работы в режиме реального времени и способную распознавать новые классы.

В виду особенностей задачи, классификатор должен обладать следующими отличительными свойствами:

— классификация по ограниченному набору признаков, основанному на свойствах первых $N$ пакетах, т.к. нет возможности получения информации обо всем потоке (достигается за счет формирования соответствующей матрицы признаков);

— обнаружение новых классов, т.е. должны применяться методы кластеризации;

— высокая точность полученных результатов (за счет разработанной статической модели классификации);

— разделение на кластеры проводится на основе динамически меняющейся, не предопределенной заранее метрики. В отличие от большинства работ по кластеризации трафика, количество кластеров заранее неизвестно и не может быть вычислено.

## 4.1. Методология решения задач кластеризации

Обычно методы кластеризации применительно к сетевому трафику показывают невысокие результаты [57; 124; 125]. Но главное преимущество методов кластеризации перед методами машинного обучения «с учителем» в возможности обнаружения новых классов, не представленных в обучающей выборке.

Авторы [126] предложили использовать иерархическую кластеризацию как способ извлечения и обобщения статистической информации об активности трафика в сетях большой емкости. В [127] методы иерархической кластеризации (*AutoFocus* и *BIRCH*) на основе *IP*-адресов и других категориальных эвристик позволили создать некоторые базовые схемы трафика. Эти и другие работы подтверждают эффективность применения иерархических методов в задачах анализа сетевого трафика.

В статье [128] вместо расстояния ***Евклида*** применяют матрицу расстояний на основе ***RF*** с последующей кластеризацией методом *K-Medoids*. Исследование показало значительное улучшение результатов кластеризации на основе расстояний с использованием ***RF*** по сравнению с расстоянием ***Евклида***. Но стоит отметить, что для выполнения кластеризации методом *K-Medoids* требуется иметь данные о количестве кластеров для разбиения, что в динамически изменяющейся сети в режиме реального времени невозможно.

В [129] Маньков В.А. впервые предложил применять агломеративную кластеризацию для классификации приложения в режиме реального времени, т.к. она поддерживает режим разделения на кластеры на основе дистанции между классами и, благодаря ее структуре, существует возможность интерпретации полученных кластеров с точки зрения политики *QoS*. Агломеративная кластеризация позволяет регулировать количество кластеров в зависимости от расстояния между кластерами. Таким образом, в отличие от других распространенных методов (*K*-средних, спектральная кластеризация и т.д.), не требуется обладать знаниями о числе кластеров до кластеризации. Эта особенность позволяет не только определить оптимальное число кластеров для обучающей выборки, но, что более важно, появляется возможность вводить новые классы в уже существующую модель автоматически, регулируя лишь расстояние между кластерами.

### 4.1.1. Агломеративная кластеризация

**Агломеративная кластеризация (*Agglomerative clustering*)** [130] является одним из видов иерархических структур и может быть представлена в виде дендрограммы, позволяющей упорядочивать данные «снизу-вверх»: первоначально каждый элемент выборки считается отдельным кластером, по мере уменьшения расстояния между кластерами, объекты объединяются между собой, образуя новые кластеры (Рисунок 4.1).

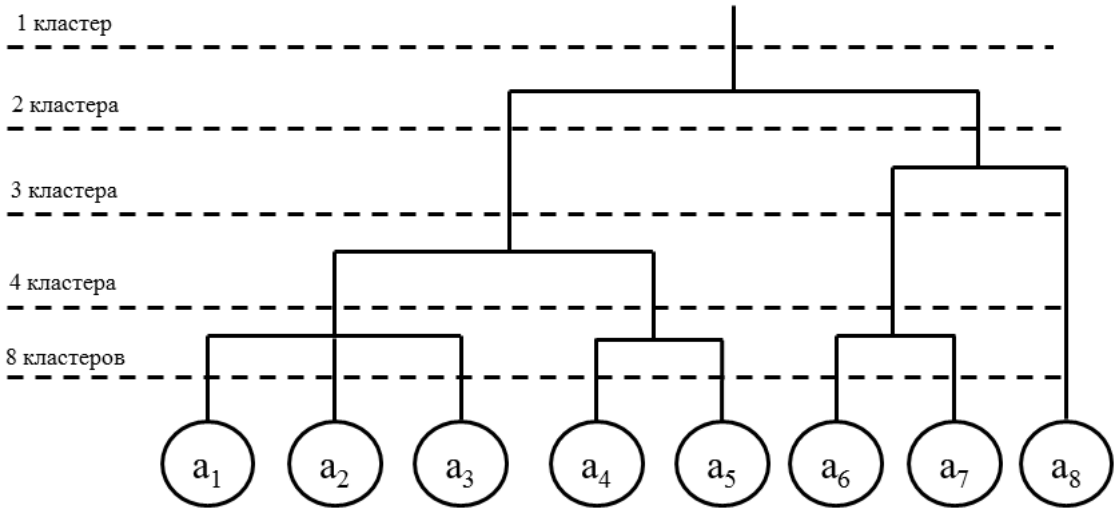



Рисунок 4.1 - Пример дендрограммы из восьми выборок

На каждом этапе работы агломеративной кластеризации подсчитывается расстояние между образовавшимися кластерами с помощью формулы Ланса – Уильямса [131]:

$$R(W,S) = \alpha_U \cdot R(U,S) + \alpha_V \cdot R(V,S) + \beta \cdot R(U,V) + \gamma \cdot |R(U,S) - R(V,S)|, (4.1)$$

где $U, V$ – два кластера, объединяющиеся на этом шаге в один кластер $W = U \cup V$,

$S$ – любой кластер, отдельно стоящий от $W$;

$R$ - расстояние между кластерами;

$\alpha_U, \alpha_V, \beta, \gamma$ – числовые параметры, которые рассчитываются с помощью следующих методов:

1. Метод одиночной связи, ***single*** *linkage* (4.2) при $\rho$ - функция расстояния:

$$R^{single}(W,S) = \min_{w \in W,\ s \epsilon S} \rho(w,s), \tag{4.2}$$

$$\alpha_U = \frac{1}{2}, \qquad \alpha_V = \frac{1}{2}, \qquad \beta = 0, \qquad \gamma = -\frac{1}{2}.$$

2. Метод полной связи, *complete linkage* (4.3):

$$R^{complete}(W,S) = \max_{w \in W,\ s \epsilon S} \rho(w,s), \tag{4.3}$$

$$\alpha_U = \frac{1}{2}, \qquad \alpha_V = \frac{1}{2}, \qquad \beta = 0, \qquad \gamma = \frac{1}{2}.$$

3. Метод средней связи, *average linkage* (4.4):

$$R^{average}(W,S) = \frac{1}{|W||S|} \sum_{w \in W} \sum_{s \epsilon S} \rho(w,s), \tag{4.4}$$

$$\alpha_U = \frac{|U|}{|W|}, \qquad \alpha_V = \frac{|V|}{|W|}, \qquad \beta = 0, \qquad \gamma = 0.$$

4. Метод Уорда, *Ward linkage* (4.5), обычно не используется при работе с нестандартной матрицей расстояний:

$$R^{Ward}(W,S) = \frac{|S||W|}{|S|+|W|} \rho^2 \left( \sum_{w \in W} \frac{w}{|W|} \sum_{s \epsilon S} \frac{s}{|S|} \right), \tag{4.5}$$

$$\alpha_U = \frac{|S|+|U|}{|S|+|W|}, \qquad \alpha_V = \frac{|S|+|V|}{|S|+|W|}, \qquad \beta = \frac{-|S|}{|S|+|W|}, \qquad \gamma = 0.$$

При этом для нахождения начального расстояния $R$ между двумя выборками (потоками) тоже могут применяться различные методы. Среди стандартных подходов в работах чаще всего выделяют расстояние ***Евклида*** и ***Манхэттена***. Подробнее о формировании матрицы расстояний в Подпункте 4.1.2.

### 4.1.2. Формирование матрицы расстояний

Матрицей расстояний называется матрица, в которой указаны расстояния между каждой парой объектов (потоков). Рассматриваются стандартные способы подсчета расстояний (расстояние ***Евклида*** и ***Манхэттена***) и нестандартные способы, основывающиеся на предварительно рассчитанной матрице расстояний (методы ***Random Forest, Extremely Randomized Trees*** [132; 133] и ***Random Trees Embedding*** [134]). С точки зрения контроля над построением матрицы расстояний, их можно условно разделить на две группы:

1. Неконтролируемый расчет расстояний: ***Евклидово*** расстояние, ***Манхэттенское*** расстояние, ансамбль совершенно случайных деревьев (***Random Trees Embedding***). При таком способе подсчета расстояний, метки классов не используются, расстояния рассчитываются только на основе матрицы признаков.
2. Контролируемый расчет расстояний: «случайный лес», «чрезвычайно случайные деревья». При таком методе выборка разбивается на классы методами машинного обучения «с учителем», а далее считается расстояние между элементами выборки.

#### *4.1.2.1. Расстояние Евклида (euclidean)*

Пусть матрица признаков имеет вид (3.19). Для построения матрицы расстояний на ее основе, для каждой пары потоков $A_t = [a_{t1},\ a_{t2}, \dots, a_{tm}]$ и $A_r = [a_{r1},\ a_{r2}, \dots, a_{rm}]$, где $r, t \in [1; k]$ рассчитывается расстояние ***Евклида*** (4.6):

$$\rho^{euclidean}\ (\mathrm{A}_t, \mathrm{A}_r\ ) = \sqrt{\sum_{j=1}^{m} \left(a_{tj} - a_{rj}\right)^2}. \qquad (4.6)$$

#### *4.1.2.2. Расстояние Манхэттена (manhattan)*

Матрица расстояний на основе расстояния ***Манхэттена*** рассчитывается аналогично, с помощью формулы (4.7):

$$\rho^{man\ hattan}\ (\mathrm{A}_t, \mathrm{A}_r\ ) = \sum_{j=1}^{m} \left|a_{tj} - a_{rj}\right|. \qquad (4.7)$$

На Рисунке 4.2 изображена схема расчета матриц расстояния на основе стандартных методов.


Рисунок 4.2 - Матрица расстояний на основе расстояний ***Евклида*** и ***Манхэттена***

#### *4.1.2.3. Матрица расстояний на основе деревьев решений*

Для получения предварительно рассчитанной матрицы расстояний используют алгоритмы на основе деревьев решений: ***Random Forest*** (***RF***, «случайный лес»), ***Extremely Randomized Trees*** (***ET***, «чрезвычайно случайный лес») и ***Random Trees Embedding*** (***RTE***, «совершенно случайные деревья»).

Основные отличия ***Extremely Randomized Trees*** [132] от ***Random Forest***:

— для ***RF*** строятся подвыборки из начальных данных, а для ***ET*** – вся выборка;

— ***RF*** выбирает признак для разделения в узле оптимальным образом, ***ET*** – случайно. Оптимизация для ***ET*** достигается за счет выбора лучшего из подмножеств при разделении.

Таким образом, ***ET*** уменьшает систематическую ошибку и дисперсию.

***Random Trees Embedding*** представляет собой алгоритм построения «случайного леса» в режиме неконтролируемого обучения, т.е. не использует информацию о классах обучающей последовательности.

Процесс создания матрицы расстояний на основе деревьев решений (Рисунок 4.3) можно представить в виде следующей последовательности действий:

1. Расчет матрицы признаков $k \times m$ на основе статистических характеристик потока, где $k$ – количество потоков, а $m$ – количество признаков.

2. Построение леса из $n$ деревьев на основе матрицы признаков обучающей последовательности, для ***ET*** и ***RF*** проходит в контролируемом режиме.

3. Присвоение каждому потоку индекса листа, на котором он оказался на каждом из деревьев. В результате этого шага получается матрица листьев $n \times k$.

4. Создание матрицы попарной близости $k \times k$, где между каждой парой потоков указывается общее количество листьев, на которых они оказались вместе, среди всех деревьев. Процесс сравнения листьев проводится с помощью *One-Hot Encoding* [135] (представление каждого состояния с помощью одного триггера) и произведения закодированной матрицы индексов и ее же в транспонированном виде.

5. Получение матрицы расстояний путем нормирования матрицы близости относительно максимального значения и вычитания ее значений из единицы.

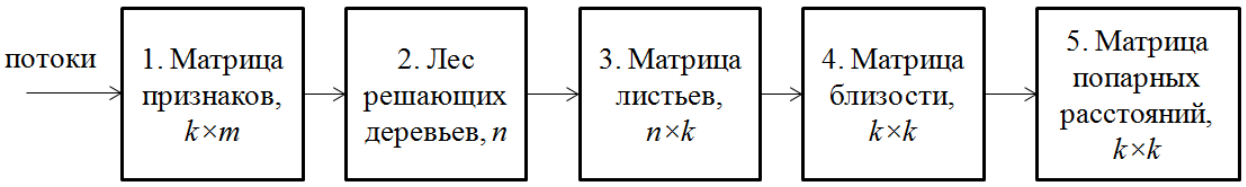



Рисунок 4.3 - Построение матрицы расстояний на основе деревьев решений

Таким образом, результатом этой последовательности действий будет являться матрица попарных расстояний $k \times k$, где ее элементы распределены в пределах [0;1] и 1 – максимальная дистанция между потоками, а диагональ состоит из 0.

### 4.1.3. Методы оценки результатов кластеризации

Пусть $X = \{x_1, x_2, \dots, x_k\}$ множество потоков $k$, которые разбиты на классы $U = \{U_1, U_2, \dots, U_R\}$ (истинные значения) и кластеры $V = \{V_1, V_2, \dots, V_C\}$ (предсказанные значения). Тогда для множеств $U$ и $V$ можно составить таблицу непредвиденных обстоятельств (таблицу сопряженности, Таблица 4.1, [136]):

Таблица 4.1. Таблица непредвиденных обстоятельств

| $U \backslash V$ | $V_1$ | $V_2$ | … | $V_C$ | $\sum$ |
|---|---|---|---|---|---|
| $U_1$ | $n_{11}$ | $n_{12}$ | … | $n_{1C}$ | $a_1$ |
| $U_2$ | $n_{21}$ | $n_{22}$ | … | $n_{2C}$ | $a_2$ |
| … | … | … | … | … | … |
| $U_R$ | $n_{R1}$ | $n_{R2}$ | … | $n_{RC}$ | $a_R$ |
| $\sum$ | $b_1$ | $b_2$ | … | $b_C$ | |

Здесь $n_{ij}$ - число потоков, которые попали одновременно во множество $U$ и $V$.

Энтропию $H$ для множеств $U$ и $V$ можно рассчитать по формуле (4.8):

$$H(U) = -\sum_{i=1}^{R} \frac{a_i}{k} \log \frac{a_i}{k}, \tag{4.8}$$

$$H(V) = -\sum_{j=1}^{C} \frac{b_j}{k} \log \frac{b_j}{k}.$$

Условную энтропию $H$ для множеств $U$ и $V$ можно рассчитать по формуле (4.9):

$$H(U|V) = -\sum_{i=1}^{R} \sum_{j=1}^{C} \frac{n_{ij}}{k} \cdot log \frac{\frac{n_{ij}}{k}}{\frac{b_j}{k}}. \tag{4.9}$$

Индекс общей информации $I$ можно рассчитать по формуле (4.10):

$$I(U,V) = \sum_{i=1}^{R} \sum_{j=1}^{C} \frac{n_{ij}}{k} \cdot log \frac{\frac{n_{ij}}{k}}{\frac{a_i b_j}{k^2}}. \tag{4.10}$$

Оценка результатов кластеризации проводится на основе Таблице 4.1 с помощью следующих наиболее распространенных параметров [137].

#### *4.1.3.1. Согласованный индекс Рэнда (Adjusted Rand Index)*

***Adjusted Rand Index (ARI, согласованный индекс Рэнда*** [138]***)*** измеряет согласие двух выборок: предсказанной и истинной, игнорирует перестановки и нормализуют все значения в пределах [-1; 1], где -1 – плохая маркировка, 0 - случайные метки, а 1 – лучший показатель соответствия.

Пусть $N_{00}$ - количество пар потоков, оказавшихся в одном кластере на предсказанной выборке ($V$) и в одном классе на истинной выборке ($U$);

$N_{01}$ - количество пар потоков, которые находятся в одном классе в $U$, но в разных кластерах во множестве $V$;

$N_{10}$ - количество пар потоков, которые находятся в разных классах в $U$, но в одинаковых кластерах во множестве $V$;

$N_{11}$ - количество пар потоков, которые находятся в разных классах в $U$ и в разных кластерах во множестве $V$.

Иногда $N_{00}$ и $N_{11}$ называются индексами согласия между множествами $U$ и $V$, а $N_{01}$ и $N_{10}$ – индексами разногласия.

***Согласованный индекс Рэнда (ARI)*** рассчитывается по формуле (4.11):

$$ARI\ (U,V) = \frac{2(N_{00}N_{11} - N_{01}N_{10})}{(N_{00} + N_{01})(N_{01} + N_{11}) + (N_{00} + N_{10})(N_{10} + N_{11})}. \quad (4.11)$$

#### *4.1.3.2. Скорректированная взаимная информация*

***Adjusted Mutual Information (AMI, скорректированная взаимная информация*** [139]***)*** измеряет меру сходства между множествами $U$ и $V$.

Ожидаемая взаимная информация для множеств $U$ и $V$ определяется по формуле (4.12):

$$E\{I(U,V)\} = \sum_{i=1}^{\mathrm{R}} \sum_{j=1}^{\mathrm{C}} \sum_{n_{ij}=\max(a_i+b_j-k)}^{\min(a_i,b_j)} \frac{\frac{n_{ij}}{k}\log\left(\frac{kn_{ij}}{a_i b_j}\right)\left(a_i!\, b_j!\,(k-a_i)!\,(k-b_j)!\right)}{k!\, n_{ij}!\,(a_i - n_{ij})!\,(b_j - n_{ij})!\,(k - a_i - b_j + n_{ij})!}. \quad (4.12)$$

Тогда скорректированная взаимная информация ($AMI$) рассчитывается по формуле (4.13):

$$AMI(U,V) = \frac{I(U,V) - E\{I(U,V)\}}{max\big(H(U),H(V)\big) - E\{I(U,V)\}}. \quad (4.13)$$

#### *4.1.3.3. Однородность (Homogeneity)*

***Однородность (Homogeneity)*** показывает, что в каждом кластере находятся объекты, принадлежащие только одному классу (4.14):

$$Homogeneity = 1 - \frac{H(U|V)}{H(U)}. \quad (4.14)$$

#### *4.1.3.4. Полнота кластеров (Completeness)*

По ***полноте кластеров (Completeness)*** можно определить долю выборки одного класса, принадлежащей одному кластеру (4.15):

$$Completeness = 1 - \frac{H(V|U)}{H(V)}. \quad (4.15)$$

Таким образом, ***однородность*** характеризует ошибки первого рода, а ***полнота кластеров*** - ошибки второго рода.

#### *4.1.3.5. V-мера (V-measure)*

***V-мера (V-measure)*** - это среднее гармоническое между ***однородностью*** и ***полнотой кластеров*** (4.16):

$$V_{measure} = 2 \cdot \frac{Homogeneity \cdot Completeness}{Homogeneity + Completeness}. \quad (4.16)$$

Все три характеристики (***однородность***, ***полнота кластеров*** и их ***гармоническое среднее***) нормированы в пределах [0; 1], где 1 соответствует лучший результат.

## 4.2. Результаты кластеризации применительно к сетевому трафику

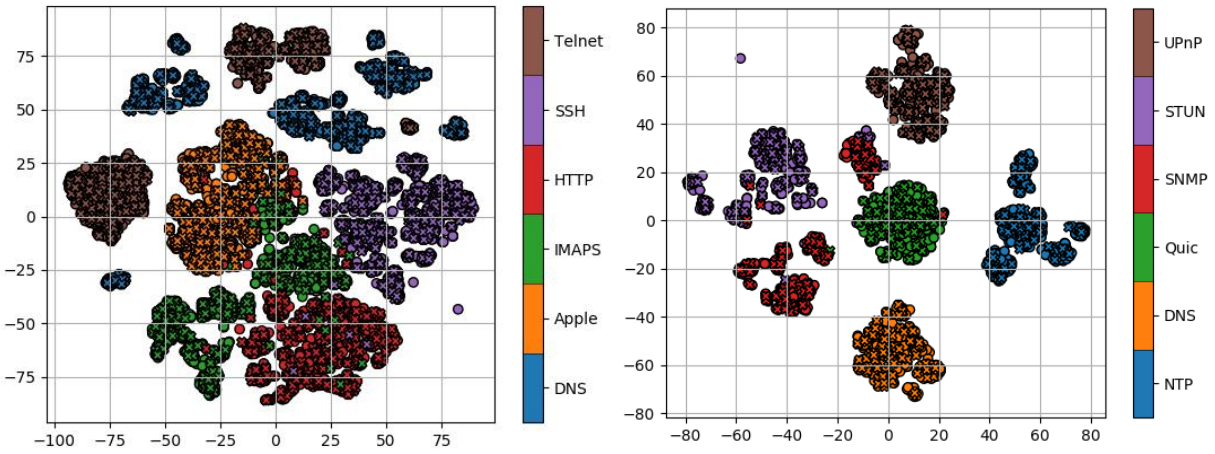
Экспериментальная часть исследований этого раздела базируется на наборе данных Подраздела 2.2. Для работы над моделью кластеризации созданы по две группы для TCP и UDP–потоков. Первая группа используется в экспериментах по

первоначальной кластеризации, выборов подходящих параметров и настроек. Вторая группа участвует в экспериментах для проверки возможности модели обнаруживать новые классы. Первая группа состоит из 1200 потоков каждого из *TCP*–приложений: *Telnet, SSH, HTTP, IMAPS, Apple, DNS*; вторая группа – по 1200 потоков: *RTMP*, *IMAP*, *SMTP* и *Skype*. Первая группа *UDP*-потоков (по 200): *UPnP, STUN, SNMP, Quic, DNS* и *NTP*, а вторая группа: *IPsec, NetBIOS* и *Apple*.

### 4.2.1. Визуализация кластеров с помощью метода t-SNE

Широко распространенный в *ML* метод визуализации ***t-SNE*** (Стохастическое вложение соседей с *t*-распределением, ***t-distributed Stochastic Neighbor Embedding*** [140; 141]) позволяет понизить размерность данных и изобразить их в двухмерном пространстве таким образом, чтобы точки из одной выборки с наибольшей вероятностью оказывались ближе друг к другу, а из разных – дальше друг от друга. Метод очень сильно зависит от случайных составляющих и не является точным определением расстояний между точками, но с его помощью удается получить представление о возможности кластеризации и наборе данных.

На Рисунке 4.4 представлены результаты, полученные в результате визуализации методом ***t-SNE*** выбранных приложений. Кругами отмечены выборки обучающей последовательности, а крестиками - тестовой. Из рисунка видно, что выборка способна разбиться на кластеры, хоть и имеет пересечения в некоторых местах – например, для *TCP* - пересечение *IMAPS, HTTP* и *Apple*. Появление таких спорных зон может быть вызвано не только трудностями кластеризации, но и особенностями работы протоколов, а также возможными ошибками автоматической разметки.




а) б)

Рисунок 4.4 - Визуализация кластеров с помощью ***tSNE*** *TCP* (а) и *UDP*-приложений (б)

### 4.2.2. Расчет матрицы расстояний

Для исследования способов расчета матриц расстояний (4.1.2) проводится два эксперимента: зависимость оценок кластеризации (4.1.3) от числа кластеров и оценка кластеризации в зависимости от общего числа потоков для разных матриц расстояний.

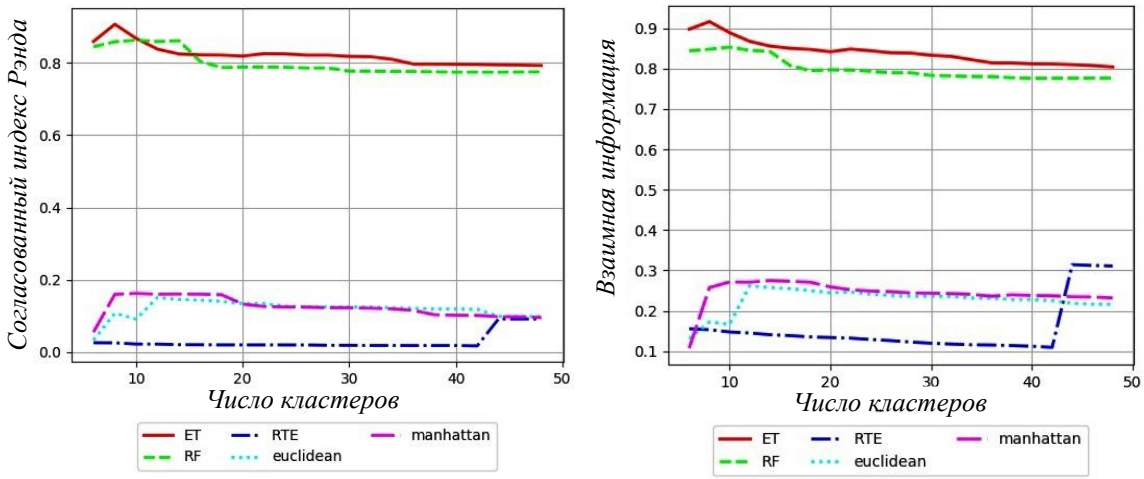
Контролируемые способы построения матрицы расстояний (***ET*** и ***RF***) использовали обучающий массив для построения деревьев, а тестовый массив строился на основе уже имеющихся деревьев без использования меток классов. Неконтролируемые методы (***RTE***, ***Евклидово*** и ***Манхэттенское*** расстояние) не применяли метки классов при создании матриц расстояний, но для возможности сравнения с ***ET*** и ***RF*** результаты кластеризации тестового и обучающего массивов показаны отдельно друг от друга.

#### *4.2.2.1. Зависимость результатов кластеризации от числа кластеров для разных матриц расстояний*

На основе матриц расстояний, полученных пятью различными способами (***ET***, ***RF***, ***RTE***, ***Евклидово*** и ***Манхэттенское*** расстояние) выполнялась агломеративная кластеризация с числом кластеров, меняющимся от 6 до 45. По

результатам кластеризации проводилась ее оценка методами ***ARI, AMI, полнотой кластеров,*** их ***одородностью*** и ***V-мерой***. Кластеризация выполнялась с помощью библиотеки *Scikit-learn* [15] языка программирования *Python*.

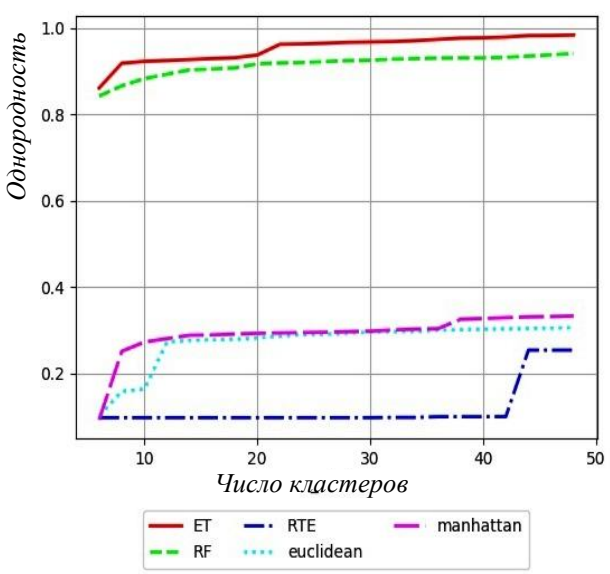
На Рисунке 4.5 изображены зависимости ***ARI*** и ***AMI*** от числа кластеров для *TCP*-приложений. Видно, что лучшие показатели ***ARI*** и ***AMI*** для обучающей последовательности *TCP*-приложений достигаются методом ***ET*** (близки к 1). ***ARI*** для ***RTE*** находится на 0-01, а ***AMI*** на 0,12-0,30, следовательно, методу ***RTE*** не удалось верно провести кластеризацию. При расчете расстояний ***Манхэттена*** и ***Евклида*** результаты не намного лучше и находятся в пределах 0,15-0,25, что также показывает непригодность этих методов в данной ситуации.



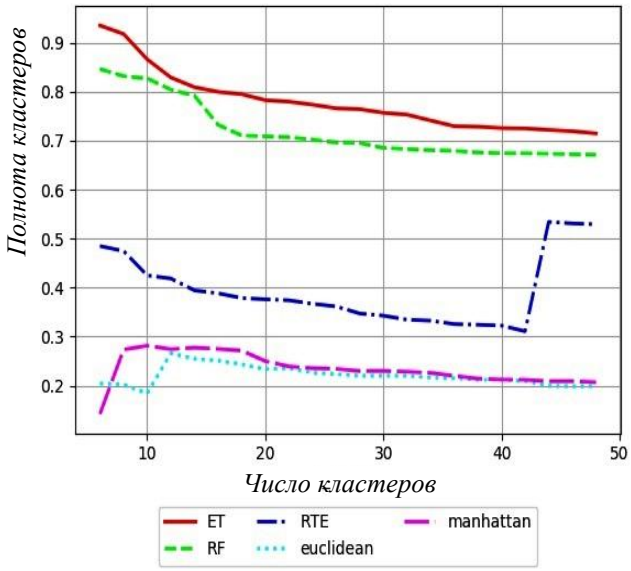

а) б)

Рисунок 4.5 - Зависимости с***огласованного индекса Рэнда (ARI)*** (а) и ***скорректированной взаимной информации (AMI)*** (б) от числа кластеров для *TCP*-потоков

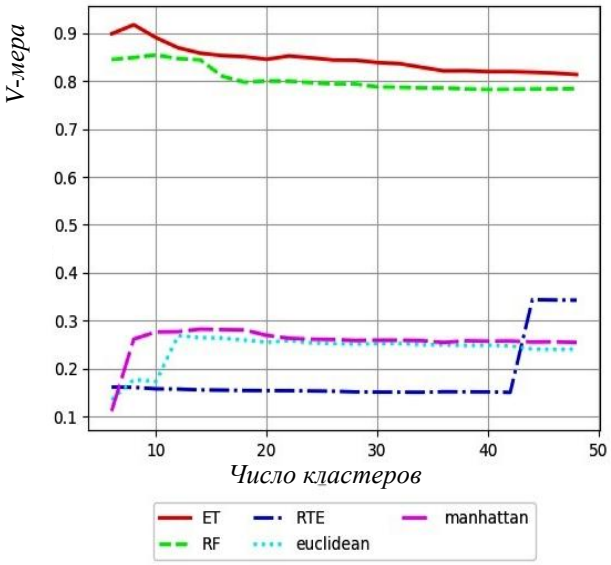
Все те же выводы подтверждаются графиками ***однородности, полноты кластеров*** и ***V-меры*** (Рисунок 4.6). Такое значительное различие в результатах кластеризации методами ***ET, RF*** и методами ***RTE***, расстояния ***Евклида*** и ***Манхэттена*** объясняется наличием контроля построения кластеров у ***ET*** и ***RF***.




а)




б)

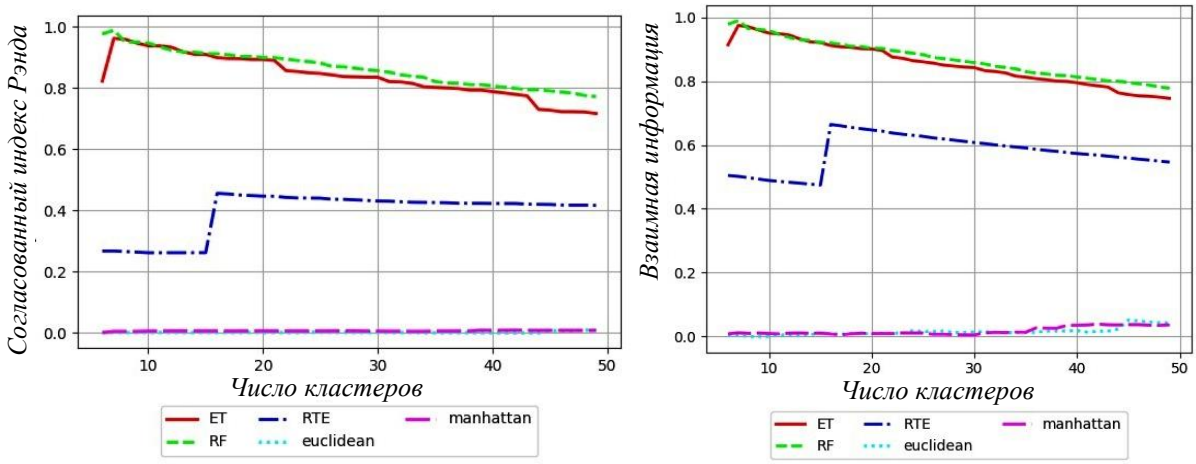



в)

Рисунок 4.6 - Зависимости оценок ***однородности*** (а), ***полноты кластеров*** (б) и ***V-меры*** (в) от числа кластеров для *TCP*-приложений

Для *UDP*-приложений методы ***ET*** и ***RF*** показывают результаты ***ARI*** и ***AMI*** в пределах 0,95-1 (Рисунок 4.7) при небольшом числе кластеров. Аналогично ситуации с кластеризацией *TCP*-приложений, методы, основанные на расстояниях ***Манхэттена*** и ***Евклида***, не справляются с задачей и находятся в пределах 0-0,3. ***ARI RTE*** для *UDP* несколько выше, чем для *TCP* - на уровне 0,3-0,4, а ***AMI*** - 0,55-

0,65. По мере увеличения числа кластеров ***ARI*** и ***AMI*** уменьшались для ***RF*** и ***ET*** (с 1 до 0,75), для остальных методов немного увеличивались (0-0,2).




а) б)

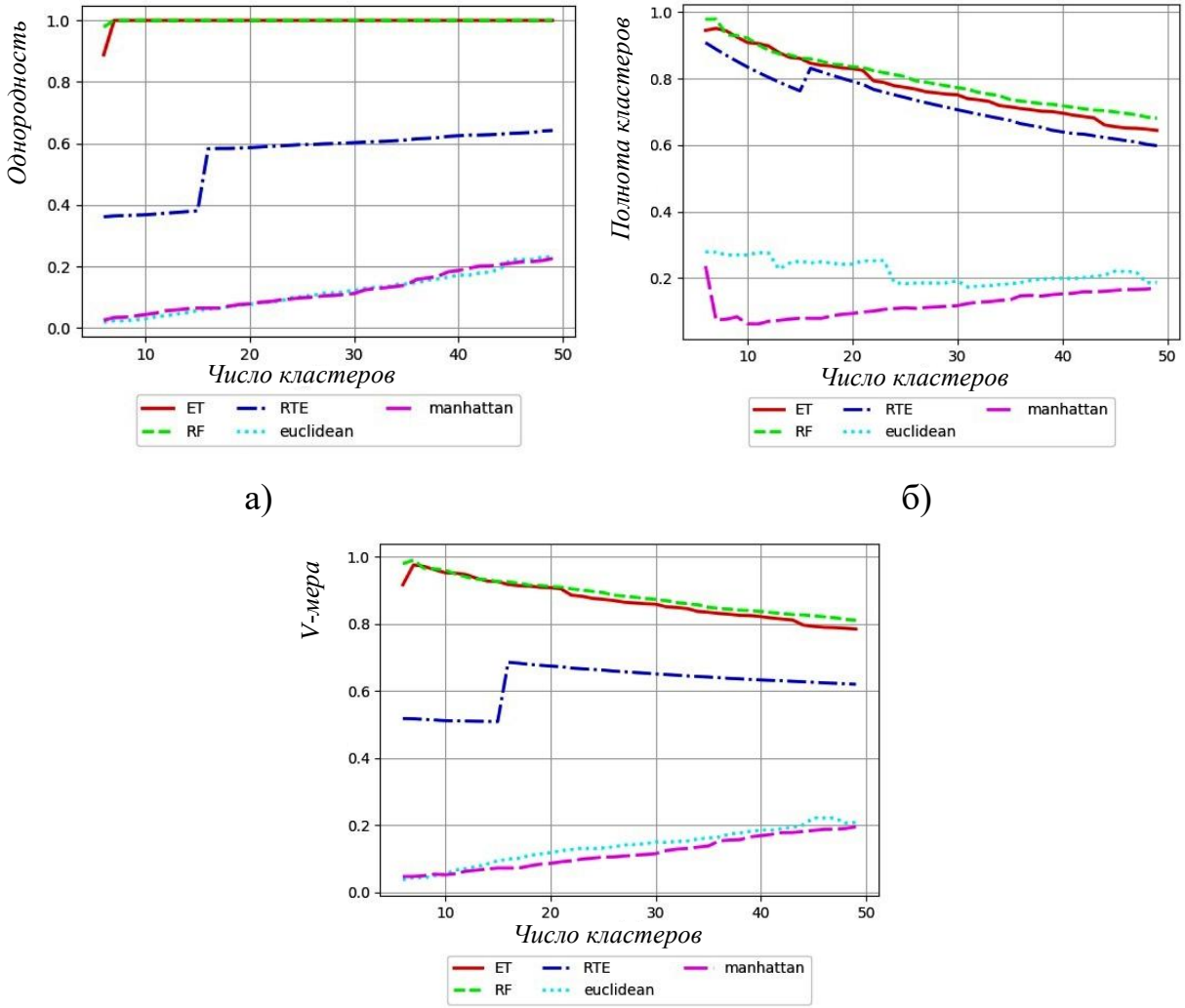
Рисунок 4.7 - Зависимость с***огласованного индекса Рэнда (ARI)*** (а) и ***скорректированной взаимной информации (AMI)*** (б) от числа кластеров для *UDP*-приложений

***Однородность, полнота кластеров*** и ***V-мера*** также в этом случае выше, чем для *TCP* (Рисунок 4.8). Такой результат объясняется самим составом кластеров - в случае *UDP* кластеры более различимы друг от друга, чем в случае *TCP* (Рисунок 4.6).

Показатели ***однородности*** ожидаемо начали возрастать, а ***полноты кластеров*** - уменьшаться. Избыточное число кластеров нежелательно применять в модели, т.к. оно вызывает ситуации, когда для одной выборки выделяется один кластер. Кроме того, избыточное число кластеров сложно поддается интерпретации с точки зрения методов управлению сетью.

Несмотря на то, что бесконтрольные методы иногда тоже показывают четкое разделение по кластерам - например, ***RTE*** определяет 43 кластера для *TCP* (Рисунок 4.5-4.6) и 16 для *UDP* (Рисунок 4.7-4.8) - значение основных характеристик кластеризации остается на очень низком уровне, что не позволяет уверенно их использовать при кластеризации. Так, ***согласованный индекс Рэнда***

при ***RTE*** для *TCP* – 12%, а для *UDP* – 50%, в то время как методы ***ET*** и ***RF*** показывают значения на уровне 95-100%.



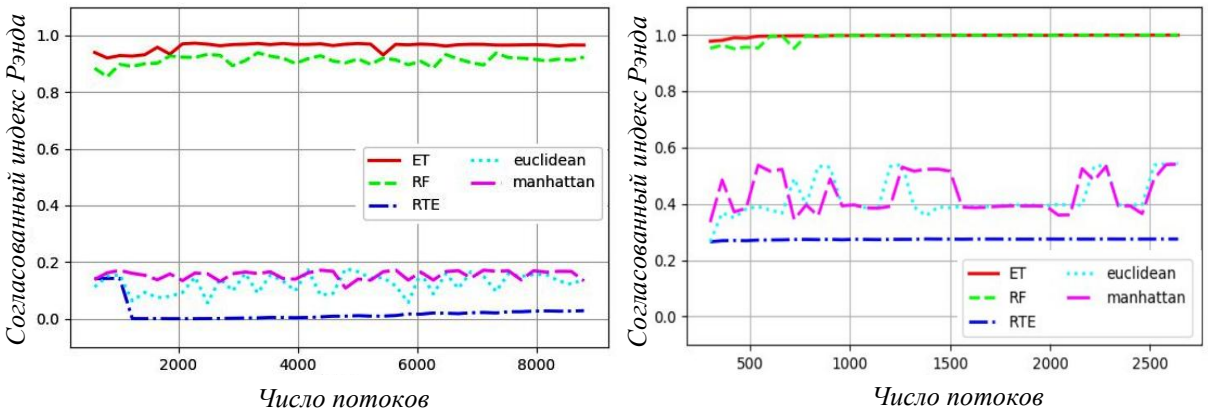

а) б)

в)

Рисунок 4.8 - Зависимости оценок ***однородности*** (а), ***полноты кластеров*** (б) и ***V-меры*** (в) от числа кластеров для *UDP*-приложений

#### *4.2.2.2. Зависимость результатов кластеризации от общего числа потоков для разных матриц расстояний*

На Рисунке 4.9 показана оценка результатов кластеризации приложений в зависимости от общего числа потоков. Ввиду слабой зависимости представлены только графики ***ARI***. Для *TCP*-приложений общее число потоков варьируется от 600 до 9000, а для *UDP* - от 200 до 2700. Число кластеров в этом эксперименте оставалось фиксированным: 8 для *TCP* и 7 для *UDP*-потоков.




а) б)

Рисунок 4.9 - Зависимость с***огласованного индекса Рэнда (ARI)*** от числа потоков для *TCP* (а) и *UDP*-кластеров (б)

Методы ***ET*** и ***RF*** показывают устойчивые результаты по мере увеличения числа потоков для каждого приложения свыше 600 для *TCP* и 150 для *UDP*. Это говорит о хорошей различимости кластеров друг от друга этими методами. Бесконтрольные методы показывают некоторые колебания результатов, но в виду их низкой точности, применение данных методов в модели неоправданно.

### 4.2.3. Результаты динамической кластеризации

#### *4.2.3.1. Набор данных TCP-приложений*

На Рисунке 4.10 построен график зависимости числа кластеров от минимального нормированного расстояния между кластерами и представлены результаты кластеризации в зависимости от числа получившихся кластеров. Из графика видно, что чем меньше минимальная допустимая дистанция между кластерами, тем больше кластеров образуется. Число кластеров уменьшается неравномерно, несмотря на равномерное увеличение дистанции.

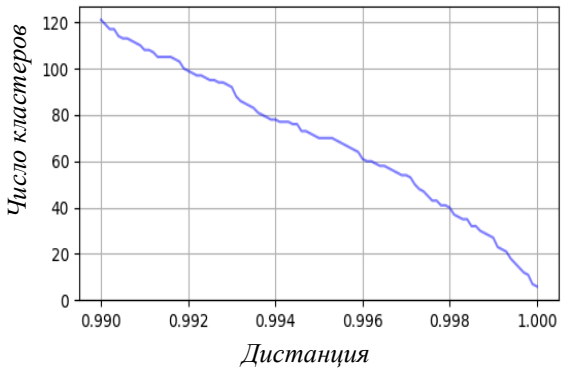
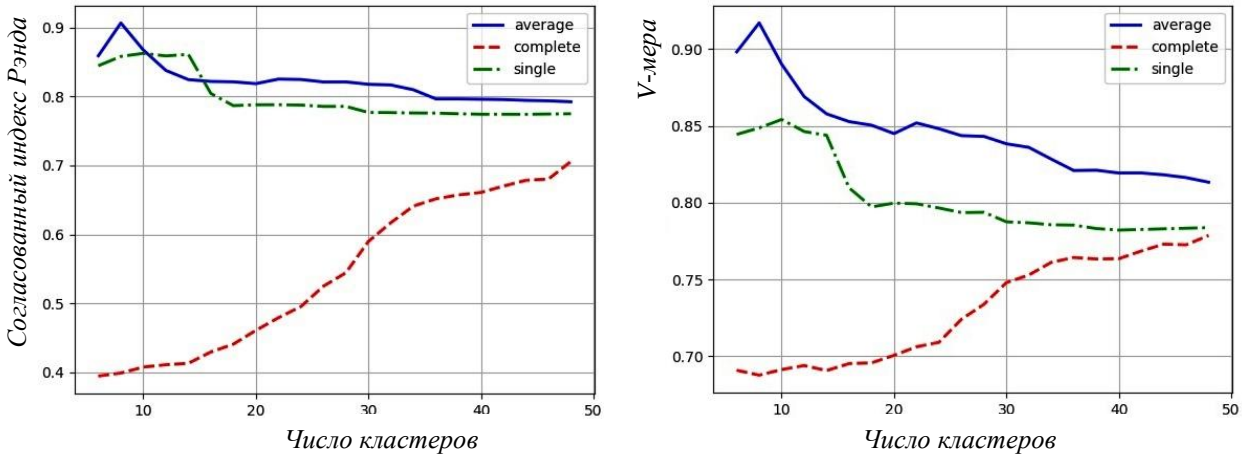



Рисунок 4.10 - Зависимость числа кластеров от выбранной минимальной дистанции между ними

Результаты ***ARI*** и ***V-меры*** показывают положительные результаты, близкие к 0,9-1,0 при числе кластеров 6-8 (Рисунок 4.11). ***V-меры*** с увеличением числа кластеров падает, т.к. теперь выборки одних и тех же классов принадлежат разным кластерам. С точки зрения классификации трафика, это не всегда является ошибкой, т.к. одному и тому же приложению может соответствовать различное поведение трафика, что подтверждается Рисунком 4.4. Следовательно, для начальной работы алгоритма нужно выбрать число кластеров больше, чем число приложений в базе данных.




а) б)

Рисунок 4.11 - Зависимость с***огласованного индекса Рэнда (ARI)*** (а) и ***V-меры*** (б) от числа кластеров для *TCP*-приложений

Из Рисунков 4.11 также следует, что лучшей мерой к расчету дистанции между кластерами для *TCP*-приложений является метрика средней связи ***average*** (4.2). На Рисунке 4.12 показано значение гармонического среднего между ***ARI*** и ***V-мерой***. Исходя из результатов эксперимента, для первичного обучения модели было выбрано 8 кластеров (1 *DNS*, 1 *Telnet*, 1 *SSH*, 1 *Apple*, 2 *IMAPS* и 2 *HTTP*) с минимальной относительной дистанцией между кластерами 0,9996, матрица непредвиденных обстоятельств для нее изображена в Таблице 4.2.

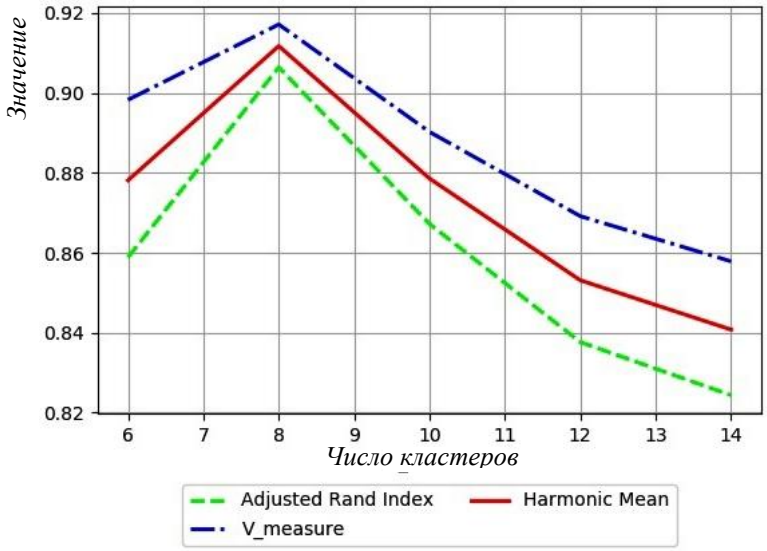



Рисунок 4.12 - Гармоническое среднее (***Harmonic Mean***) между ***ARI*** и ***V-мерой*** для *TCP*-потоков

Таблица 4.2. Таблица непредвиденных обстоятельств для *TCP*-приложений

| *U\V* | | Кластеры | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ | $V_7$ | $V_8$ |
| Классы | *Telnet* | 1200 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | *SSH* | 0 | 1200 | 0 | 0 | 0 | 0 | 0 | 0 |
| | *HTTP* | 0 | 0 | 1051 | 146 | 1 | 0 | 2 | 0 |
| | *IMAPS* | 0 | 0 | 4 | 8 | 1133 | 48 | 7 | 0 |
| | *Apple* | 0 | 0 | 0 | 5 | 0 | 2 | 1193 | 0 |
| | *DNS* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1200 |

Для проверки способности модели обнаруживать новые виды классов, на вход модели поочередно добавлялись по 300 потоков с каждого из приложений *RTMP*, *IMAP*, *SMTP* и *Skype*. После каждого добавления замерялось число кластеров. На Рисунке 4.13 изображен график зависимости числа образовавшихся кластеров от состава приложений. Видно, что при добавлении каждого нового класса добавляется новый кластер, кроме случаев с добавлением *IMAP*. Это объясняется наличием небольшого кластера *HTTP*, который объединился с выборками *IMAP* (Таблица 4.3, Рисунок 4.14).

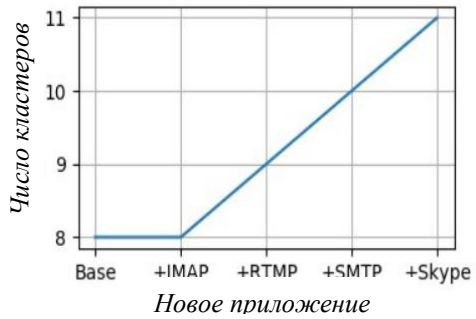



Рисунок 4.13 - Образование новых кластеров при добавлении новых классов для *TCP*-приложений

Таблица 4.3. Обновленная таблица сопряженности для *TCP*-потоков

| *U\V* | | Старые кластеры | | | | | | | | Новые кластеры | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ | $V_7$ | $V_8$ | $V_9$ | $V_{10}$ | $V_{11}$ |
| Старые классы | Telnet | 1200 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | SSH | 0 | 1200 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | HTTP | 0 | 0 | 1051 | 146 | 1 | 0 | 2 | 0 | 0 | 0 | 0 |
| | IMAPS | 0 | 0 | 4 | 8 | 1133 | 48 | 7 | 0 | 0 | 0 | 0 |
| | Apple | 0 | 0 | 0 | 5 | 0 | 2 | 1193 | 0 | 0 | 0 | 0 |
| | DNS | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1200 | 0 | 0 | 0 |
| Новые классы | IMAP | 0 | 0 | 0 | 298 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| | RTMP | 0 | 0 | 3 | 1 | 1 | 0 | 0 | 0 | 295 | 0 | 0 |
| | SMTP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 300 | 0 |
| | Skype | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 300 |

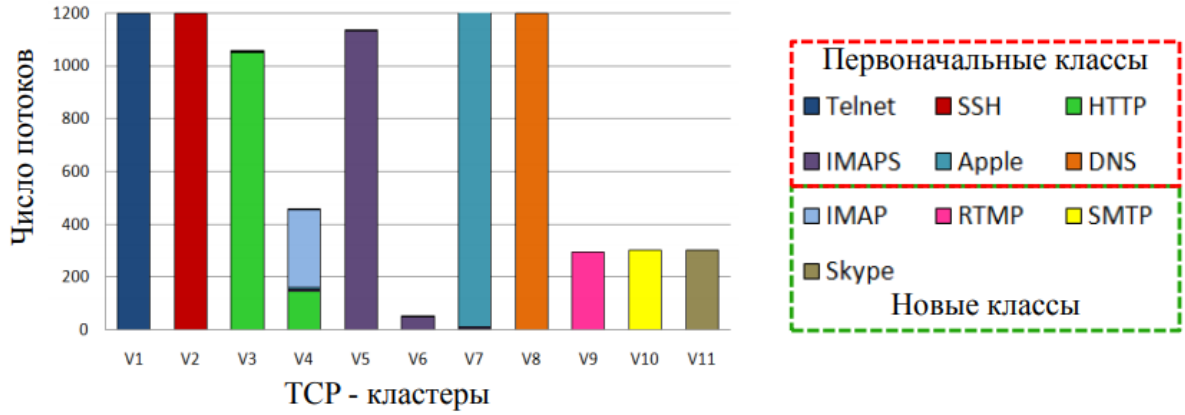



Рисунок 4.14 - Качественный состав кластеров при добавлении классов *TCP*-приложений

Разбиение на кластеры проходит на основании минимальной дистанции между кластерами. Важно отметить, что при каждой новой процедуре кластеризации количество классов не должно уменьшаться. Ситуация с уменьшением числа кластеров может возникнуть при добавлении в базу данных кластера, занимающего промежуточную позицию между некоторыми из существующих кластеров. Таким образом, дистанция между ними становится меньше первоначально выбранной и несколько кластеров объединяются в один. Для предотвращения этой проблемы, метки существующих выборок фиксируются, а в спорных ситуациях дистанция, выбранная для разделения на кластеры, уменьшается.

#### *4.2.3.2. Набор данных UDP-приложений*

На Рисунке 4.15 построен график зависимости числа кластеров от минимального нормированного расстояния между кластерами и представлены результаты кластеризации в зависимости от числа получившихся кластеров. Число кластеров, как и в случае с *TCP*, уменьшается неравномерно, несмотря на равномерное увеличение дистанции.

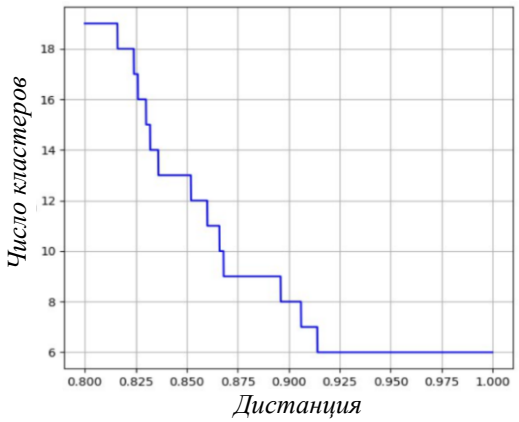
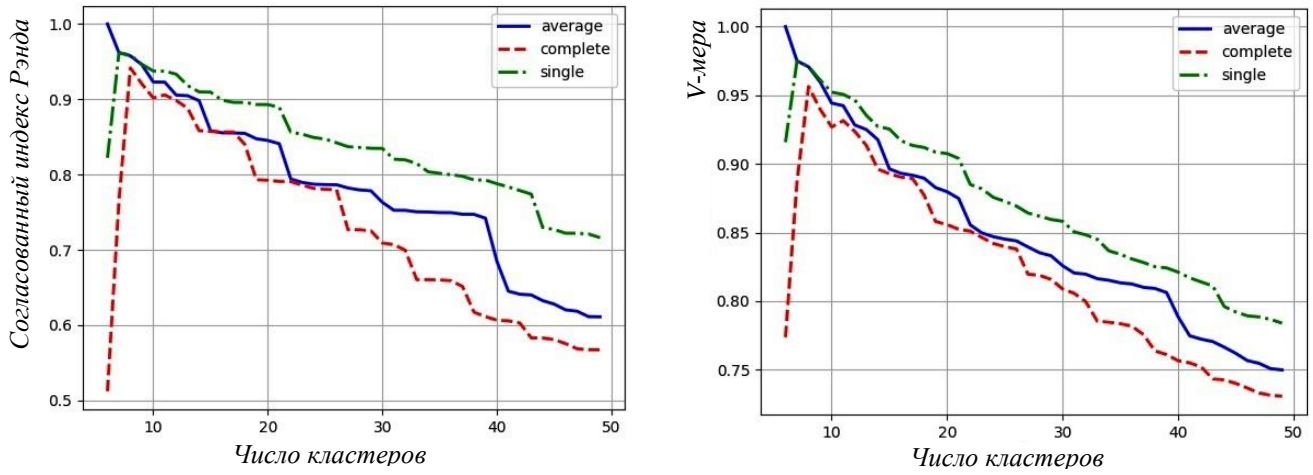



Рисунок 4.15 - Зависимость числа кластеров от выбранной минимальной дистанции между ними для *UDP*-приложений

Результаты кластеризации для *UDP* аналогичны результатам для *TCP* (Рисунок 4.16). Лучшие результаты, близкие к 0,9-1, модель принимает при числе кластеров, близким к 6-8, и мере расстояния ***single***.




а) б)

Рисунок 4.16 - Зависимость с***огласованного индекса Рэнда (ARI)*** (а) и ***V-меры*** (б) от числа кластеров для *UDP*-приложений

На Рисунке 4.17 показано значение гармонического среднего между величинами ***ARI*** и ***V-мерой***, которое достигает своего максимума при 7 кластерах и минимальной относительной дистанцией между кластерами 0,912. Начальный состав кластеров представлен в Таблице 4.4.

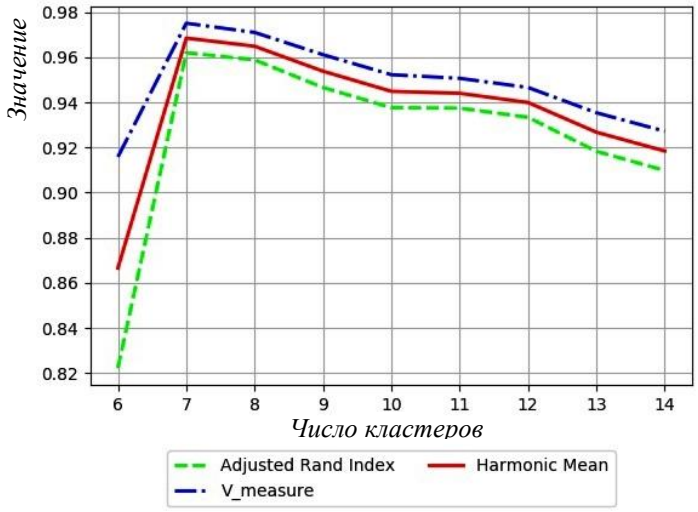



Рисунок 4.17 – Гармоническое среднее (***Harmonic Mean***) между ***ARI*** и ***V-мерой*** для *UDP*-потоков

Для проверки способности модели обнаруживать новые виды классов, на вход модели поочередно добавлялись по 200 потоков с каждого из приложений *IPsec*, *NetBIOS* и *Apple*. В Таблице 4.5 представлена матрица непредвиденных обстоятельств. Видно, что при добавлении каждого нового класса добавляется новый кластер (Рисунок 4.18).

Таблица 4.4. Таблица непредвиденных обстоятельств при 7 кластерах для *UDP*-приложений

| *U\V* | | Кластеры | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ | $V_7$ |
| Классы | *UPnP* | 198 | 2 | 0 | 0 | 0 | 0 | 0 |
| | *STUN* | 0 | 200 | 0 | 0 | 0 | 0 | 0 |
| | *SNMP* | 0 | 0 | 199 | 1 | 0 | 0 | 0 |
| | *Quic* | 0 | 0 | 0 | 55 | 145 | 0 | 0 |
| | *NTP* | 0 | 0 | 0 | 0 | 0 | 200 | 0 |
| | *DNS* | 0 | 0 | 0 | 0 | 0 | 0 | 200 |

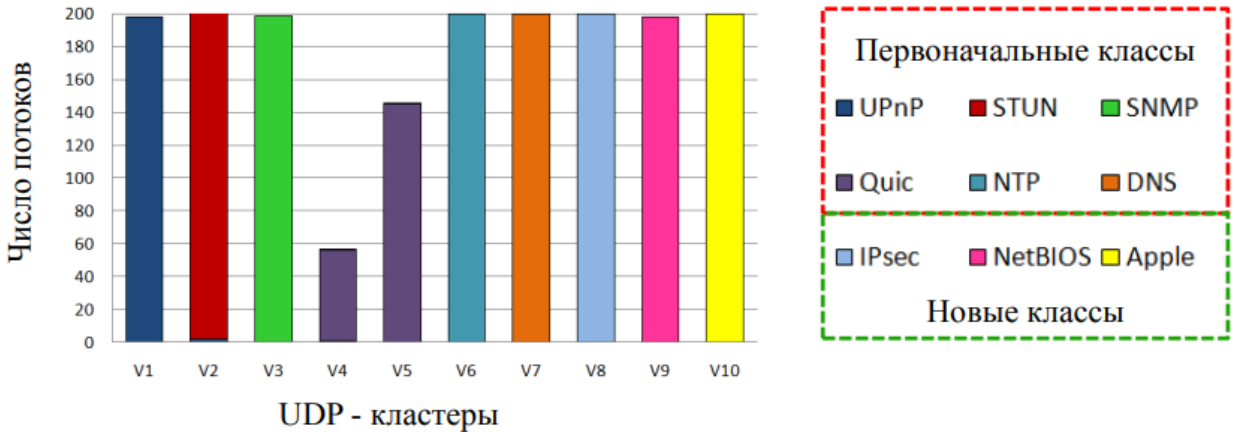



Рисунок 4.18 - Качественный состав кластеров при добавлении классов *UDP*-приложений

Таблица 4.5. Таблица непредвиденных обстоятельств при 10 кластерах для *UDP*-приложений

| *U\V* | | Старые кластеры | | | | | | | Новые кластеры | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ | $V_7$ | $V_8$ | $V_9$ | $V_{10}$ |
| Старые классы | *UPnP* | 198 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | *STUN* | 0 | 200 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | *SNMP* | 0 | 0 | 199 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | *Quic* | 0 | 0 | 0 | 55 | 145 | 0 | 0 | 0 | 0 | 0 |
| | *NTP* | 0 | 0 | 0 | 0 | 0 | 200 | 0 | 0 | 0 | 0 |
| | *DNS* | 0 | 0 | 0 | 0 | 0 | 0 | 200 | 0 | 0 | 0 |
| Новые классы | *IPsec* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 200 | 0 | 0 |
| | *NetBIOS* | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 198 | 0 |
| | *Apple* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 200 |

## 4.3. Функциональная модель классификации трафика

Итоговая модель классификации трафика в режиме реального времени с возможностью добавления новых классов представлена на Рисунке 4.19.

В блоке сбора информации (1) для поступающего в сеть пакета записывается время прихода и его размер. После этого, пакеты группируются в

потоки на основе 5-*tuple* (*IP*-адреса, порты источника и назначения и протокол). Для того чтобы система работала в режиме реального времени, учитываются только первые 10-15 пакетов потока, а результаты классификации применяются для последующих пакетов [70; 87]. Подробнее о блоке сбора информации, адаптированного под матрицу признаков для классификации, описано в пятом разделе.

Далее, для каждого потока рассчитывается матрица признаков, описанная во втором разделе (2). В качестве признаков выступают статистические свойства потока: длина каждого пакета, межинтервальное время и характеристики, рассчитанные на их основе (медиана, дисперсия, СКО, минимальное и максимальное значения, битовая и пакетная скорости) [94].

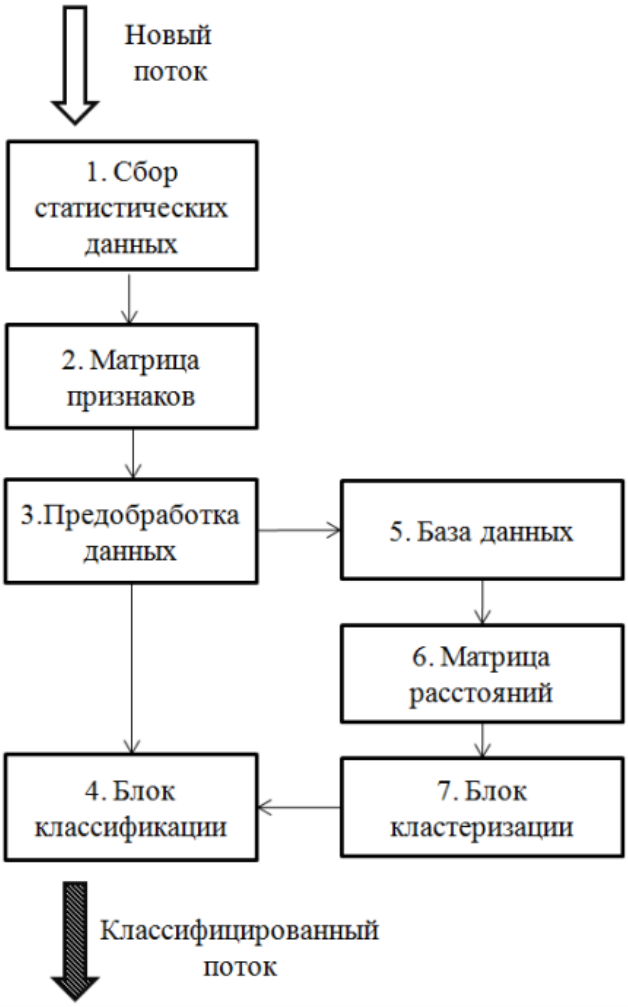



Рисунок 4.19 - Модель классификации трафика в режиме реального времени с возможностью добавления новых классов

В блоке предварительной обработки данных (3) происходит работа с выбросами, нормализация и масштабирование полученной матрицы признаков [111].

В следующем, блоке классификации (4), с помощью *Random Forest* или *XGBoost* проводится классификация трафика по известным для модели классам. В том случае, если приложение является новым и незнакомым для модели, на этом этапе могут возникнуть некоторые ошибки, т.к. методы обучения «с учителем» способны классифицировать выборки только по известным для него классам. Такое допущение в модели сделано осознанно, т.к. дальнейшая работа по управлению трафиком основана на известных классах. Тем не менее, блок классификации (4) представляет наиболее близкий класс для пришедшего нового потока и как следствие, для него применяются соответствующие правила *Traffic Engineering (TE)*.

В то время как блок классификации (4) решает, к какому классу отнести активный поток, а блок *TE* применяет правила по управлению трафиком в режиме реального времени, копия информации о новом потоке из блока (3) отправляется в блок (5). Здесь хранится база данных по всем известным потокам. Каждая выборка представляется своими координатами в *N*-мерном пространстве в соответствии с определенными признаками.

После построения матрицы расстояний между выборками (6) [129], в блоке кластеризации (7) проводится разделение выборок на кластеры с помощью агломеративной кластеризации [142].

После анализа некоторого количества новых потоков и появления дополнительных кластеров, блок классификации (4) обучается новым классам, а инструменты *TE* определяют новые правила для управления потоками. Подобный подход к построению модели классификатора позволяет использовать скорость классификации методами машинного обучения «с учителем» (*RF* и *XGBoost*) и возможность идентифицировать новые потоки методами обучения «без учителя» (Агломеративная кластеризация).

Для работы модели требуется около 200-250 Мбит ОЗУ (для *Intel Core Processor, Haswell, no TSX, IBRS 2394.454 MHz CPU*), *Python 3.7*, режим обновления модели занимает около 0,1-0,12 мс, а режим классификации ≈ 2 мкс (при одновременной классификации до 100 потоков).

## Выводы по четвертому разделу

1. Создана модель кластеризации трафика, высокая точность которой (0,9-1) достигается за счет использования предварительно рассчитанной матрицы расстояний на основе метода ***Extremely Randomized Trees.*** Стандартные методы расчета матрицы расстояний, такие как расстояние ***Евклида*** и ***Манхэттена*** неприменимы к разработанной матрице признаков.
2. Для кластеризации на основе разработанной модели необходимо около 600 потоков каждого приложения для *TCP* и 150 для *UDP*.
3. Создана не имеющая аналогов в открытых исследованиях модель классификатора трафика, полученная за счет объединения точности и скорости работы методов «обучения с учителем» для работы в режиме реального времени с возможностью добавления новых классов за счет методов «обучения без учителя» и уточнения существующих кластеров. Несмотря на то, что работа ориентирована на задачи *SDN*-сети, результаты работы могут быть использованы и в других сетях.

Результаты четвертого раздела обсуждались на конференциях *CSDEIS2020* и *28-й FRUCT* (2021) и опубликованы в статьях [129; 142].

# Раздел 5. Разработка программного обеспечения для сбора статистической информации о сетевых пакетах

В большинстве работ по классификации трафика методами машинного обучения (Приложение Б) задача сбора статистической информации не рассматривается вовсе или в качестве инструментов мониторинга используется утилита *tcpdump*. Алгоритм сбора данных о пакетах для многих задач может быть не так важен, особенно, когда речь идет о классификации трафика вне режима реального времени или когда классификация подразумевается не для работы в реальной сети. Но в режиме реального времени и в формате ограниченной информации о пакетах (10-15) требуется обладать необходимыми инструментами для сбора статистической информации. Разработанный метод должен:

— работать в режиме реального времени;

— полностью предоставлять необходимую информацию для разработанной матрицы признаков (Раздел 2);

— не загружать контроллер и сетевые элементы избыточной служебной и нагрузочной информацией.

## 5.1. Программное обеспечение

### 5.1.1. Краткие сведения о P4

В настоящее время в *SDN*-сетях широкое распространение получил протокол *OpenFlow* [143 - 145], за счет которого возможно значительно упростить плоскость управления по сравнению с традиционными сетями [146, 147]. Это стандартизированный протокол, для взаимодействия контроллера с коммутаторами, позволяющий устанавливать *Flow Table* (таблицы потоков), получать данные о статистике потоков и т.д. Концепция *Flow Table* модели

*OpenFlow* основывается на простых принципах: если поля поступившего пакета соответствуют записи для потока, то выполняется заранее определенное действие. Недостатками такого подхода являются следующие факты:

— эволюция протокола требует добавления все большего числа полей соответствия;

— открытыми остаются вопросы совместимости между различными вендорами (различные варианты *OpenFlow, Netconf, JSON, XML*);

— ограниченные возможности для программирования коммутатора. Возможности приложений заранее определены, требования сети не позволяют повлиять на расширение функционала коммутаторов.

Иной подход к организации сети предложили *P. Bosshart, D. Daly , G. Gibb* и др. в статье *«P4: Programming Protocol-independent Packet Processors»*, 2014 г [67]. *P4* – это и язык программирования, разработанный для программирования коммутаторов в сетях *SDN*; и название специфических коммутаторов *P4*; и название интерфейса, с помощью которого взаимодействуют плоскость данных и плоскость управления.

Концепция *P4* предлагает для каждой сети использовать индивидуальный подход, с учетом требований сети.

Для начала работы сети выбирается одна из архитектурных моделей коммутаторов *P4* либо создается своя собственная, так называемая «поведенческая модель» (*«behavioral-model»*) [148]. Она должна определять ключевые программируемые параметры, включать в себя весь процесс обработки пакетов и являться неким шаблонов для составления рабочей программы. Одна из наиболее распространенных моделей, используемая в т.ч. и в этой работе, носит название *simple_switch* [149], о которой более подробно будет изложено ниже.

По выбранному шаблону для коммутаторов создается код программы на *P4*, который должен отвечать требованиям и нуждам сети. С помощью выбранного компилятора, код с *P4* компилируется в файл формата *json* и в дальнейшем может быть загружен в плоскость передачи данных сети *SDN*.

Плоскость управления в данном случае будет аналогична плоскости управления в сетях с использованием протокола *OpenFlow*. С ее помощью загружаются или удаляются данные таблиц потоков, проводится мониторинг и т.д. Функционал этой плоскости зависит от выбранной архитектурной модели *P4* и созданного кода для плоскости передачи данных.

Таким образом, *P4* позволяет:

— определять синтаксический анализ и процесс обработки пакетов с помощью контроллера, менять настройки и возможности уже в ходе работы коммутатора и таким образом легко масштабировать сеть;

— быть независимыми от протоколов: коммутаторы не привязываются ни к каким конкретным протоколам, благодаря чему могут применяться любые протоколы, в т.ч. *OpenFlow*, либо не применяться вообще никакие, в таком случае управление проводиться только за счет программирования на *P4*;

— пакет-функциональная обработка пакетов не зависит от спецификаций базового оборудования, компилятор учитывает все возможности, перечисленные в коде программы на *P4*, и перекомпилирует их на плоскость передачи данных.

### 5.1.2. Особенности моделирования с помощью сетевого эмулятора Mininet

Сетевой эмулятор *Mininet* [150] создает реалистичную виртуальную сеть, на которой запущено реальное ядро, коммутаторы и код приложения на одной машине с помощью одной команды.

Так как *Mininet* может взаимодействовать с реальными сетями, прост в управлении через *CLI* (и *API*), позволяет развёртывать сети на реальном оборудовании, то его часто используют для разработок, обучения и исследований.

Еще одним преимуществом эмулятора *Mininet* является возможность работать с *SDN*-контроллерами.

Базовые функции *Mininet*:

— выбор встроенной топологии с помощью набора команд либо создание собственной на языке *Python*;

— возможность выполнения простейших сетевых команд или операций на уровне mininet-консоли;

— получение информации о сети;

— управление хостами с помощью эмулятора терминала *XTerm*;

— работа с удаленным контроллером;

— эмуляция различных коммутаторов в сети, в работе используются *P4*-коммутаторы.

### 5.1.3. Настройка лабораторного стенда

Лабораторный стенд (Рисунок 5.1) представляет собой хостовую машину под операционной системой *Centos 7*, на которой установлена виртуальная машина *p4* под управлением *Ubuntu 16.04 LTS*. Доступ в интернет может быть осуществлен с помощью *Open vSwitch* [151].

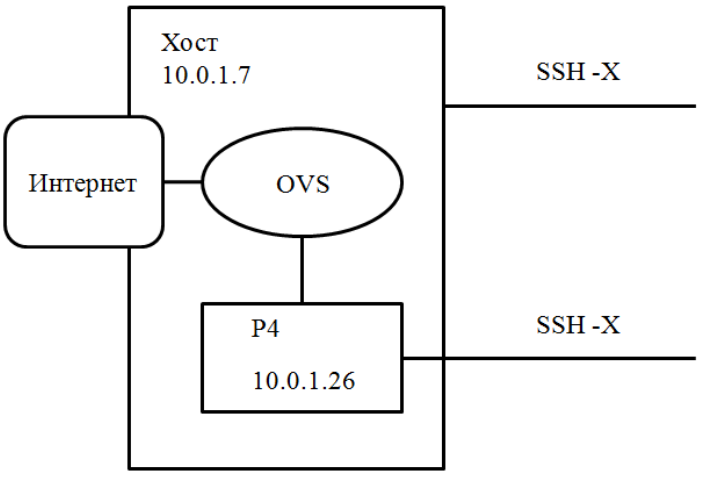



Рисунок 5.1 - Схема лабораторного стенда

Параметры виртуальной машины *p4*:

— *IP*-адрес: *10.0.1.26*;

— доступ: *SSH*;

— гипервизор: *KVM*;

— архитектура: *x86_64*;

— память: 48000 *MiB*;

— диск: 60 *GiB*

— *CPU*: 4;

— наличие графического интерфейса: *нет*;

— операционная система: *Ubuntu 16.04 LTS*.

## 5.2. Модель сбора статистической информации о пакетах

На Рисунке 5.2 изображена модель сбора статистической информации о пакетах сетевых потоков с помощью *P4*-коммутатора. На уровне *SDN*-контроллера находятся основные функциональные блоки классификатора трафика (Рисунок 4.16). Полный листинг программы для *P4*–коммутатора представлен в Приложении В.

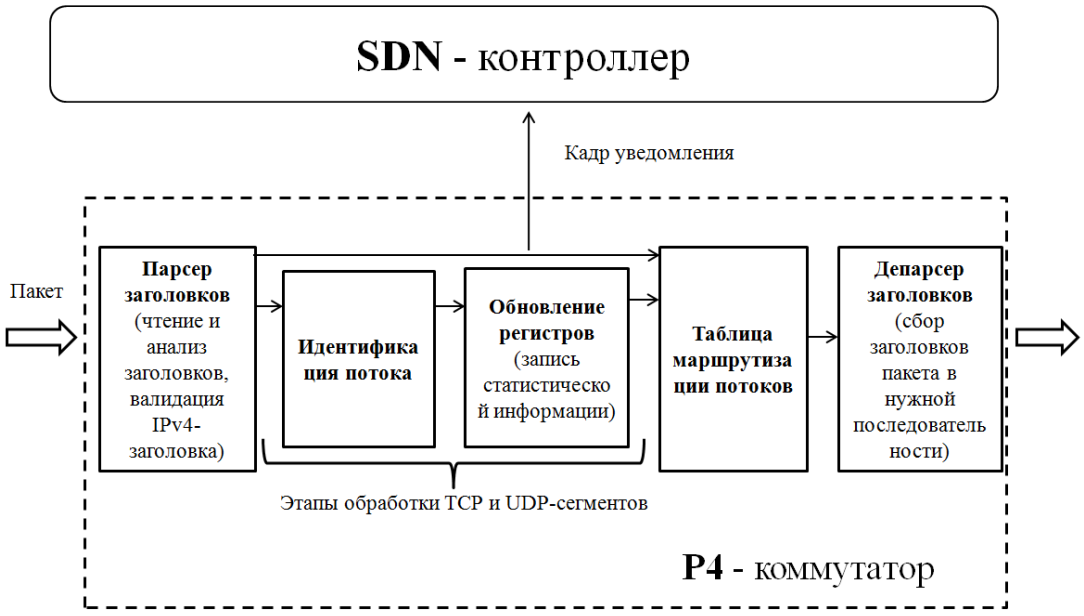



Рисунок 5.2 - Поведенческая модель simple_switch

Основные этапы обработки пакета *P4*-коммутатором:

1. **Парсер**. Заголовок пакета разбивается на его отдельные составляющие. Возможно запрограммировать парсинг любой глубины, независимо от количества заголовков, специальных меток, туннелирования и степени распространенности протокола. Благодаря тому, что все встречаемые в конкретной сети заголовки заранее определены кодом программы, существует возможность внедрения собственных протоколов и их детальный анализ с помощью коммутатора. В условиях лабораторной установки список сокращен до следующих протоколов: заголовок кадра *Ethernet II* [152], пакета *IPv4* [153], *TCP* [154] и *UDP*-сегметов [155].

У парсера существуют три основных состояния: начать (*start*), разрешить (*accept*) и запретить (*reject*). В каждом из состояний либо выполняется, либо не выполняется парсинг, после чего, согласно заранее заданным правилам обработки, парсер переходит в другое состояние, в конечном счете, завершающееся либо запретом обработки, либо ее разрешением.

Схема состояний парсера в общем виде показана на Рисунке 5.3. Предполагается, что в сети на уровне передачи данных используется только протокол *Ethernet II*, поэтому первым действием распарсивается заголовок *ethernet*. Определяется значение поля *etherType*: если оно равно *0x0800*, то парсер переходит в состояние *parse_ipv4*, если нет – то парсинг заканчивается и начинается проверка контрольной суммы.

После последовательного считывания полей заголовка *IPv4*, значение поля протокол (*ipv4.protocol*) сравнивают с зарезервированными номерами 11 и 6. Если *ipv4.protocol*=6, то парсер переходит в состояние *parse_tcp*, если 11 – в состояние *parse_udp*, в остальных случаях парсинг заканчивается и в работу вступает блок проверки контрольной суммы.

При каждом из этих двух состояний проводится парсинг заголовка пакета в соответствии с определенными ранее полями, после чего наступает следующий блок обработки пакетов.

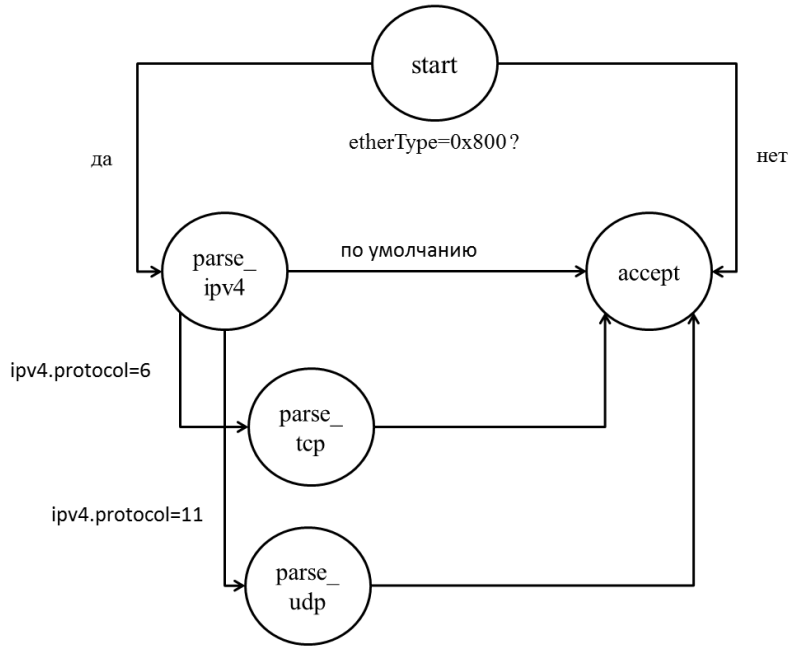



Рисунок 5.3 - Схема состояний парсера

2. **Проверка контрольной суммы** перед обработкой пакета.

3. **Идентификация потока и обновление регистров** предназначены только для *TCP* и *UDP*–пакетов и разработка этого блока является задачей пятого раздела диссертации.

Для идентификации потоков трафика и сохранения информации о них на коммутаторе, предлагается использовать особые ячейки памяти коммутатора, называемые *регистрами*. Исходя из количества и предназначений ячеек памяти, можно выделить три типа регистров: ***одиночный регистр, регистры потоков*** и ***регистры пакетов*** (Рисунок 5.4).

***Одиночный регистр*** носит название *flow_ID* и представляет собой только одну ячейку памяти. Он служит уникальным идентификатором потоков внутри коммутатора. Значения, которые в нем встречаются – это адреса ячеек памяти регистров, в которых записывается информация о соответствующем потоке. То значение, которое имеется во *flow_ID* в выбранный момент времени, показывает номера ячеек памяти, в которые будет записан следующий новый поток.

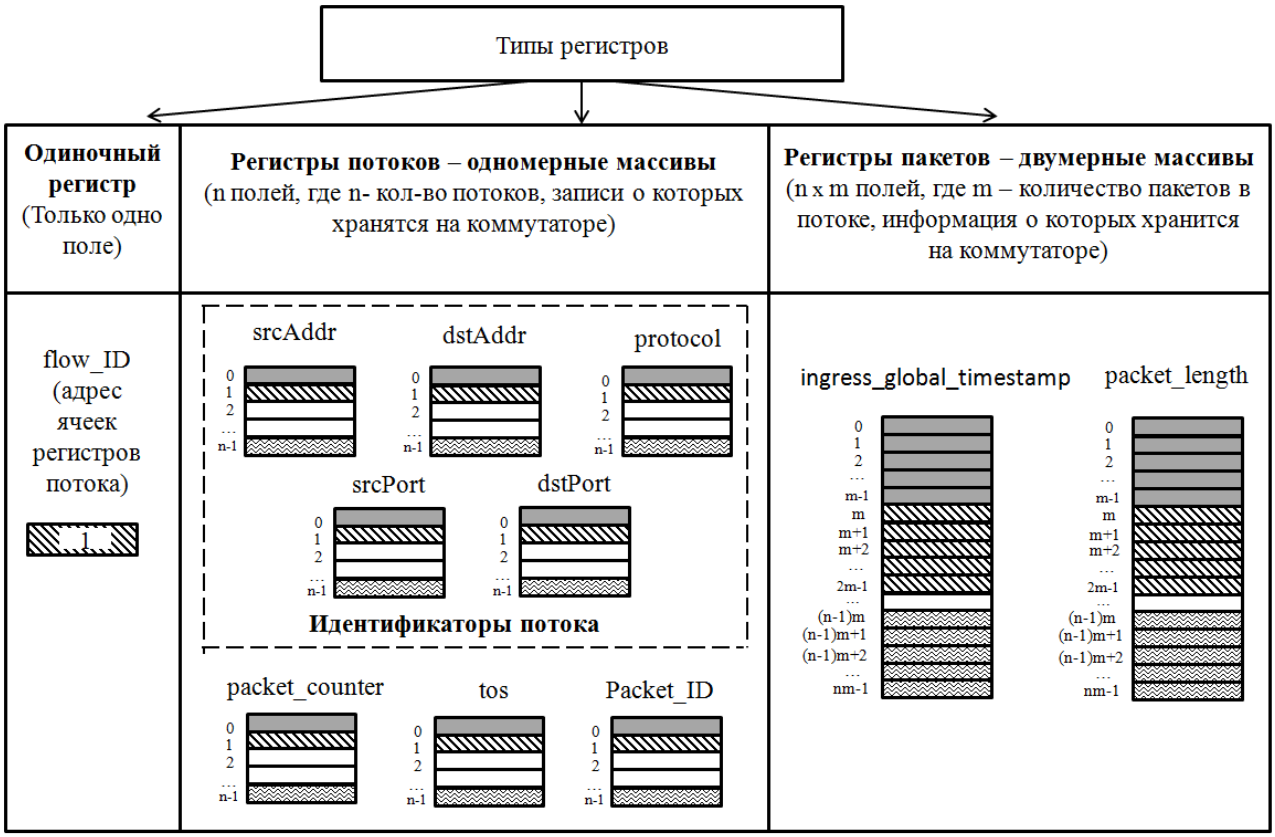



Рисунок 5.4 - Система организации памяти в P4-коммутаторе

***Регистры потоков*** – это одномерные массивы, размер которых *n* - совпадает с числом потоков, информация о которых должна храниться в памяти коммутатора. К регистрам потоков относятся: *srcAddr* (*IP*-адрес источника), *dstAddr* (IP-адрес назначения), *protocol* (протокол транспортного уровня), *srcPort* (порт источника), *dstPort* (порт назначения), *tos* (значение поле *DSCP IP*–пакета); *packet_counter* (счетчик количества пакетов потока, прошедших через коммутатор), *Packet_ID* (аналог *flow_ID* для пакетов) и *tos* (поле для маркировки *QoS*).

Поля регистров: *srcAddr* (4 байта), *dstAddr* (4 байта), *protocol* (1 байт), *srcPort* (2 байта), *dstPort* (2 байта) вместе образуют значение *5-tuple*, которое используется для идентификации потока на уровне всей сети. Поля регистра *tos* (1 байт) могут применяться не только непосредственно для обеспечения *QoS* стандартными методами, но и в качестве возможного маркера с целью

дальнейшей классификации потоков методами машинного обучения «с учителем».

***Регистры пакетов*** представляют собой двумерные массивы, организованные из одномерных, адреса ячеек в которых определяются с помощью регистров *flow_ID* и *Packet_ID*. Для целей классификации трафика методами машинного обучения были введены два вида регистров: *ingress_global_timestamp* (14 байт) как время прихода пакета на этап обработки коммутатора, выраженное в мкс и *packet_length* (10 байт) – полная длина пакета в байтах. Размер такого регистра $n \times m$ полей, где $n$ – число потоков, а $m$ –число пакетов, информацию о которых следует хранить в памяти коммутатора. Например, информация о 2-м пакете 0-го потока хранится в ячейках памяти с номером 0 в регистрах потоков (*srcAddr*[0], *dstAddr*[0] и т.д.) и в ячейках с номером 2 в регистрах пакетов (*packet_length*[2], *ingress_global_timestamp*[2]).

На этапе идентификации потоков определяется, к какому из имеющихся потоков относится пришедший пакет, при необходимости создается новый поток. Далее следует обновление регистров, при котором добавляется информация о новом пришедшем пакете. Когда из одного потока приходит 15-й по счету пакет (10-й для UDP), вся информация отправляется в *SDN*-контроллер с помощью ***кадра уведомления***. ***Кадр уведомлений*** содержит поле идентификации кадра, *5-tuple* потока, *tos* потока, *packet_length* и *ingress_global_timestamp* для первых 10-15 пакетов.

Для того чтобы получить информацию о пакете, необходимо в командной строке указать *register_read*, название регистра и номер интересующего поля. Например, *register_read reg_packet_counter 3* покажет количество пакетов, переданных в третьем потоке.

4. **Таблица маршрутизации потоков**. Задает определенные действия, связанные с таблицами потоков (Таблица 5.1), которых может быть несколько.

Таблица 5.1. Пример таблицы потоков *ipv4_lpm*

| Ключ (*IP*–адрес назначения) | Действия | Данные для действий |
|---|---|---|
| 10.0.0.10/32 | *ipv4_forward* | *MAC*-адрес назначения: *00:04:00:00:00:00* Выходной порт: *1* |
| 10.0.1.10/32 | *ipv4_forward* | *MAC*-адрес назначения: *00:04:00:00:00:01* Выходной порт: *2* |
| - | *drop* | |

Таблица потоков состоит из трех столбцов: ***ключ***, ***действие*** и ***данные для действий***. ***Ключ*** – параметр, по которому определяется соответствие или несоответствие записи потока. Поле ***ключа*** аналогично полям соответствия для протокола *OpenFlow*, но в отличие от него, в *P4* нет никаких ограничений для определения этого поля. Формат ***ключа*** должен учитываться на этапе конфигурации или перенастройки коммутатора, а само значения ***ключа*** указывается оператором в момент добавления новой записи для потока. Метаданные, присваиваемые пакету в ходе его обработки, и заголовки, извлеченные из пакета на этапе парсинга, сравниваются с ***ключом*** и в случае соответствия – выполняются действия, указанные в таблице действий.

В коде программы должны быть описаны все возможные действия для каждой из таблиц, а в ходе настройки коммутатора с плоскости управления данными (*control plane*) выбирается конкретное действие для записи.

***Данные для действий*** используются в тех случаях, когда необходимо изменить какое-либо поле из заголовка пакета либо изменить метаданные. Их формат также задается при конфигурации программного кода коммутатора, а значение определяется в момент добавления новой записи для потока.

При создании программного кода коммутатора необходимо определять действие по умолчанию, применяемое в случаях, когда посчитанного ***ключа*** нет

совпадений в таблице. А при задании записи потоков рекомендуется указывать номер приоритета записи – при совпадении ***ключа*** с несколькими строками в таблице потоков будет применяться наиболее приоритетная запись.

Помимо ***действий*** в таблицах потоков существуют действия, выполняемые непосредственно над данными.

После обработки пакета на входе, поля заголовков и метаданные могут меняться.

В условиях лабораторной установки у поступившего пакета извлекается *IP*-адрес назначения и сравнивается со значением ***ключа*** в таблице. Возможные ***действия***: пересылка пакета *ipv4_forward*, сброс пакета *drop* и бездействие *NoAction*, установленное по умолчанию.

Для ***действия*** *ipv4_forward* передаются следующие ***данные***: новый *MAC*-адрес назначения и выходной порт. При выполнении *ipv4_forward* проводятся следующие операции:

а) *MAC*-адрес источника меняется на *MAC*-адрес назначения;

б) *MAC*-адресом назначения становится новый *MAC*, указанный в таблице;

в) значение *TTL* уменьшается на 1;

г) пакет отправляется на выходной порт, указанный в таблице потоков.

Пакет по таблице *ipv4_lpm* будет обрабатываться только в том случае, если заголовок *IPv4* корректен. Корректность заголовка *IPv4* определяется с помощью поля длины заголовка, которое показывает, сколько 32-х битных строк записано в заголовке. Если значение меньше 5, то он считается некорректным и пакет сбрасывается.

Пример передачи пакета с хоста *h1* на хост *h2* через коммутатор *s1* показан на Рисунке 5.5.

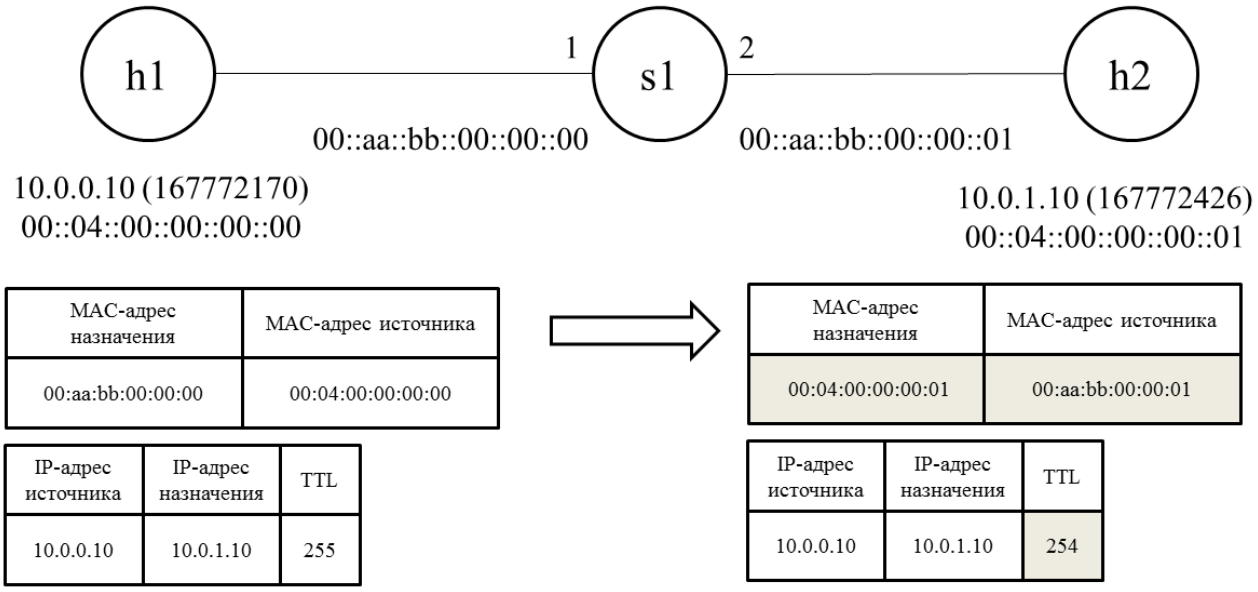



Рисунок 5.5 - Пример передачи пакета с хоста *h1* на хост *h2*

На первый порт коммутатора поступает пакет от *h1*. После парсинга, в качестве ***ключа*** для таблицы *ipv4_lpm* извлекается поле *IP*-адреса назначения, т.е. *10.0.1.10*, что соответствует второй записи в таблице потоков. К пакету должны примениться ***действия*** *ipv4_forward*. *MAC*-адрес источника меняется на *MAC*-адрес второго интерфейса коммутатора, а *MAC*–адрес назначения – *MAC* хоста *h2*, согласно таблице потоков. Метка *TTL* уменьшается на единицу. Пакет передается на второй порт.

5. **Обновление контрольной суммы** после обработки пакета.

6. **Депарсер**, позволяющий собирать все заголовки пакетов в нужной последовательности.

При написании кода основанного на поведенческой модели *simple_switch*, рекомендуется использовать специальный компилятор *p4c-bm2-ss*, который переведет код *P4* в код *json*, используемый *P4*-коммутаторами.

## 5.3. Анализ результатов классификации на основе разработанного метода сбора информации

Для оценки модели был поставлен натурный эксперимент в виртуальной сети *Mininet*. Генератором трафика выступал инструмент *D-ITG* (*Distributed Internet Traffic Generator*), который создал 1000 *UDP*-потоков от пяти разных приложений, которые поступали в сеть в случайные моменты времени по экспоненциальному закону. Три приложения имитировали работу онлайн-игр: 195 потоков *Quake3* [156], 187 потоков *CSa* (активный режим игры *Counter-Strike*) и 191 поток *CSi* [157] (неактивный режим *Counter-Strike*) и два приложения – передачу голосового трафика (кодеки *G.711.1* [158] – 207 потоков и *G.729.2* [159] – 216 потоков). Классификация проводилась методами *DT*, *GNB* и *LR* на основе матрицы признаков, сформированной по десяти пакетам.

Результаты ***точности*** классификации для разного числа потоков, используемых при обучении, представлены на Рисунке 5.6. Видно, что максимальное значение точности было получено уже при 30%-й выборке данных для обучения методом *DT*. Результаты остальных оценок при 30%-й выборке данных для обучения показаны в Таблице 5.2.

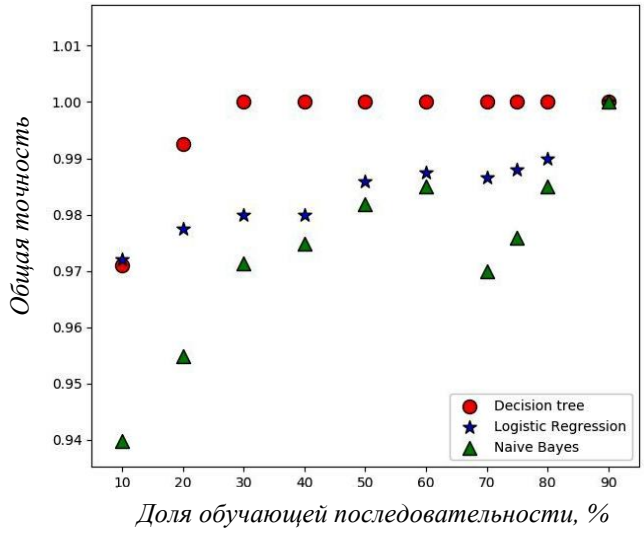



Рисунок 5.6 – ***Общая точность*** классификации в зависимости от размера данных для обучения

Таблица 5.2. Результаты классификации потоков трафика при использовании различных алгоритмов

| Классы | *Точность класса* | *Полнота* | *F1-мера* | *Точность класса* | *Полнота* | *F1–мера* | *Точность класса* | *Полнота* | *F1–мера* | Количество потоков для обучения | Количество потоков для тестирования |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *CSa* | 1,00 | 1,00 | 1,00 | 0,9 | 0,99 | 0.94 | 0,99 | 0,94 | 0,96 | 60 | 127 |
| *CSi* | 1,00 | 1,00 | 1,00 | 1,00 | 0,98 | 0,99 | 0,94 | 0,99 | 0,97 | 62 | 129 |
| *G.711.1* | 1,00 | 1,00 | 1,00 | 1,00 | 0,97 | 0,98 | 0,97 | 0,99 | 0,98 | 60 | 147 |
| *G.729.2* | 1,00 | 1,00 | 1,00 | 1,00 | 0,92 | 0,96 | 1,00 | 1,00 | 1,00 | 60 | 156 |
| *Quake3* | 1,00 | 1,00 | 1,00 | 0,96 | 1,00 | 0,98 | 0,99 | 0,97 | 0,98 | 56 | 139 |
| | *Decision Tree* | | | *Naïve Bayes* | | | *Logistic Regression* | | | | |

Согласно Рисунку 5.6 и Таблице 5.2, наилучшие результаты были получены с помощью метода «решающих деревьев». Для их достижения для обучения модели достаточно в среднем 10 первых пакетов 60 потоков. Методы «логистической регрессии» и «наивного байесовского классификатора» тоже показали высокие результаты, но безошибочной классификации с из помощью удалось добиться только при выборке данных для обучения в 90%, т.е. приблизительно при 180 потоках.

Таким образом, разработанный алгоритм сбора статистической информации о пакетах может применяться для целей классификации трафика в режиме реального времени, а классификация проводится не только с различными типами приложений, но и с различными режимами одного и того же сервиса (*CSa* и *CSi*) при наличии соответствующей тренировочной базы [10].

## 5.4. Сравнительная оценка работы методов сбора статистической информации

Для сравнительной оценки работы разработанного метода сбора статистической информации через программирование *P4*-сетей и других популярных инструментов мониторинга сетей (рассматривается на примере *tcpdump*) предлагается оценить объемы информации и долю служебной, неиспользуемой в матрице признаков информации.

Для разработанного метода передается *5-tuple*, *tos* и значения регистров *packet_length* и *ingress_global_timestamp* для первых *n* пакетов потока. Если принять за *n=15*, а размер заголовка пакета, в котором передается служебная информация, принять равным за 32, то для одного потока передается *196* байт информации, для *k* потоков, длиной более 15 пакетов, $V_{инф\ p4}=196k$. Тогда доля служебной информации при передаче результатов сбора статистической информации контроллеру, постоянна и равна $\alpha_{p4} = 0{,}23$.

При использовании *tcpdump 5-tuple* и *tos* будет передаваться для каждого пакета. Более того, собирается информация о потоках, длиной менее 15 пакетов и информация о пакетах, начиная с 16-го. Таким образом, объем информации, передаваемой с помощью *tcpdump:* $V_{инф\ tcpdump}=54S$, где *N*-количество всех пакетов всех потоков. Пусть $\{n_1, n_2 \dots, n_k\}$ – множество, в котором каждый элемент $n_i \geq 15$ - количество пакетов в *i*-м потоке, а в множестве $\{m_1, m_2 \dots, m_h\}$ каждый элемент $m_j < 15$ для *h* потоков. Тогда доля служебной информации, передаваемая в этом случае, приблизительно оценивается как (5.1):

$$\alpha_{tcpdump} = 1 - \frac{75k}{28\left(\sum_{i=1}^{k} n_i + \sum_{j=1}^{h} m_j\right)}, \qquad (5.1)$$

а разность объема передаваемой информации в байтах при использовании инструментов *tcpdump* и *P4* (5.2):

$$\Delta V_{\text{инф}} = V_{\text{инф } tcpdump} - V_{\text{инф } p4} = 56\left(\sum_{i=1}^{k} n_i + \sum_{j=1}^{h} m_j\right) - 196k. \qquad (5.2)$$

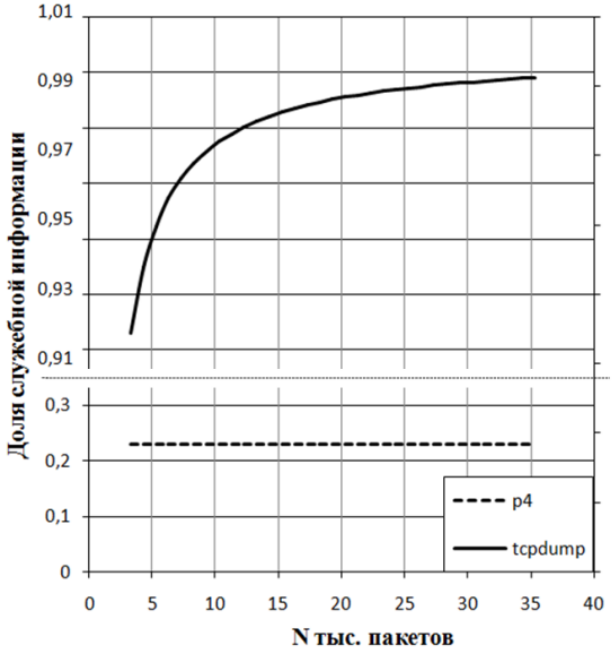



Рисунок 5.7 - Доля служебной информации в зависимости от числа проходящих через интерфейс пакетов

На Рисунке 5.7 изображен сравнительный график зависимости доли служебной информации от *N* - количества пакетов во всех потоках для *k=100* потоков. Видно, что доля служебной информации для передачи данных при использовании стандартных способов мониторинга находится в пределах от 0,91 до 1 и стремится к 1 при увеличении числа пакетов в сети. Таким образом, разработанный метод сбора статистической информации позволяет снизить объемы служебной информации в сети в 4-4,5 раза.

## Выводы по пятому разделу

1. Разработана система организации памяти в *P4*-коммутаторах для записи статистической информации о каждом пакете потока, расширяющая область применения *P4*-коммутаторов для целей обеспечения *QoS* с помощью классификации трафика методами машинного обучения.

2. Разработан алгоритм гибкого сбора статистической информации о пакете в *P4*-коммутаторах, основанный на разработанной системе организации памяти, позволяющий хранить и передавать на контроллер статистическую и идентификационную информацию о любом пакете и только по мере необходимости (например, после накопления первых 10-15 пакетов), что снижает нагрузку на сеть и устройства по сравнению с традиционными вариантами полного мониторинга сетевых элементов в 4-4,5 раза.

Результаты пятого раздела представлялись на конференциях *CSDEIS2019* и *ИТММС 2019* и были опубликованы в статьях [10; 70].

## Заключение

Основные результаты диссертационной работы сводятся к следующему:

1. Обосновано применение методов машинного обучения для решения задачи динамической классификации трафика с целью обеспечения качества обслуживания в мультисервисных *SDN*-сетях.

2. Разработана матрица признаков для классификации трафика с целью поддержания *QoS*, в которой признаками являются индивидуальные статистические параметры первых 10-15 пакетов, такие как длина и межинтервальное время прихода пакета на интерфейс. Такой подход имеет два значительных отличия от общепринятых методов построения матрицы признаков. Во-первых, он не требует никакой информации о *TCP*-флагах и заголовках вышележащего уровня, что делает разработанную матрицу признаков инвариантной по отношению к типам потоков трафика. Во-вторых, набор параметров, основанный только на первых 10-15 пакетах, позволяет эффективно применять классификацию к активным потокам в режиме реального времени. Точность полученного классификатора оказалась на 10-25% выше результатов на основе других матриц признаков.

3. Разработана и исследована статическая модель классификации трафика методами машинного «обучения с учителем» на основе созданной матрицы признаков, отличающаяся от аналогичных содержанием структурных блоков и их расположением. На каждом этапе работы модели проводятся процедуры, повышающие эффективность классификации для соответствующих методов машинного обучения.

4. Прирост точности классификации для метода *XGBoost* в блоке предварительной обработки данных (*квантильная трансформация и удаление выбросов*) – 6,5-7,2%, в блоке классификатора за счет настройки гиперпараметров — 18-20%, а суммарно до 26%, и в целом точность классификации составила значение в 91%, что на 3% выше, чем точность классификации методом *Random Forest*. Эти результаты показывают перспективность применения метода *XGBoost,*

что позволяет расширить существующий набор методов классификации потоков трафика для задач поддержания *QoS*.

5. Разработана модель эффективной кластеризации трафика (*ARI* и *AMI* около 0,9-1,0), адаптированная к сформированной матрице признаков, за счет применения в процессе кластеризации матрицы расстояний, предрассчитанной на основе результатов классификации потоков методами *Extremely Randomized Trees* и *Random Forest*. Показано также, что наиболее распространенные и стандартные подходы к расчету матриц расстояний, такие как расстояния *Евклида* и *Манхэттена*, оказались непригодными к применению в таких условиях.

6. Разработан алгоритм гибкого сбора статистической информации о пакете в *P4*-коммутаторах, основанный на созданной системе организации памяти, позволяющий хранить и передавать на контроллер статистическую и идентификационную информацию о любом пакете и только по мере необходимости (например, после накопления первых 10-15 пакетов), что снижает служебную нагрузку на сеть и устройства по сравнению с традиционными вариантами полного мониторинга сетевых элементов в 4-4,5 раза.

7. Создана не имеющая аналогов в открытых исследованиях модель классификатора трафика, полученная за счет объединения точности и скорости работы методов «обучения с учителем» для работы в режиме реального времени с возможностью добавления новых классов за счет методов «обучения без учителя» и уточнения существующих кластеров. Несмотря на то, что работа ориентирована на задачи *SDN*-сети, результаты работы могут быть использованы и в других сетях.

В дальнейших исследованиях планируется разработка методов управления трафиком на основе полученных классов для оптимального использования сетевых ресурсов и обеспечения качества обслуживания потоков поступающего трафика.

# Список сокращений

**AMI** - Adjusted Mutual Information, скорректированная взаимная информация

**API** - Application Programming Interface, интерфейс прикладного программирования

**Apple** - протокол доступа к iCloud

**ARI** - Adjusted Rand Index, согласованный индекс Рэнда

**CLI** - Command Line Interface, интерфейс командной строки

**D-ITG** - Distributed Internet Traffic Generator, генератор распределенного интернет - трафика

**DNS** - Domain Name Service, система доменных имен

**DPI** - Deep Packet Inspection, глубокий анализ пакетов

**DSCP** - Differentiated Services Code Point, точка кода дифференцированных услуг

**dstAddr** - destination address, адрес назначения

**dstPort** - destination port, порт назначения

**FTP** - File Transfer Protocol, протокол передачи файлов

**HTTP** - HyperText Transfer Protocol, протокол передачи гипертекста

**IMAP** - Internet Message Access Protocol, протокол доступа к интернет - сообщениям

**IMAPS** - IMAP over SSL, IMAP поверх SSL

**MAC** - Media Access Control, управление доступом к среде

**ML**- Machine Learning, машинное обучение

**MTU** - Maximum Transmission Unit, максимальная единица передачи

**NAT** - Network Address Translation, преобразование сетевых адресов

**ODL** - OpenDayLight

**ONOS** - Open Network Operating System

**P4** - Programming Protocol-Independent Packet Processors

**POPS** - Post Office Protocol over SSL, протокол почтового отделения через SSL

**QoE** - Quality of Experience, качество восприятия

**QoS** - Quality of Service, качество обслуживания

**RTCP** - Real-Time Transport Control Protocol, протокол управления передачей в реальном времени

**RTP** - Real-time Transport Protocol, протокол передачи в реальном времени

**SDN** - Software Defined Networking, Программно-Конфигурируемая сеть

**SMTP** - Simple Mail Transfer Protocol, простой протокол передачи почты

**SNMP** - Simple Network Management Protocol, простой протокол управления сетью

**srcAddr** - source address, адрес источника

**srcPort** - source port, порт источника

**SSH** - Secure Shell, безопасная оболочка

**SSL** - Secure Sockets Layer, уровень защищённых сокетов

**STD** - standard deviation, стандартное отклонение

**STUN** - Session Traversal Utilities for NAT, утилиты прохождения сессий для NAT

**TCP** - Transmission Control Protocol, протокол управления передачей данных

**TE** - Traffic Engineering, управление трафиком

**TOS** - Type Of Service, тип сервиса

**t-SNE** - t-distributed Stochastic Neighbor Embedding, стохастическое вложение соседей с t-распределением

**TTL** - Time to Live, время жизни

**UDP** - User Datagram Protocol, протокол пользовательских датаграмм

**UPnP** - Universal Plug and Play,

**VLAN** - Virtual Local Area Network, виртуальная локальная сеть

**VPN** -Virtual Private Network, виртуальная частная сеть

**ИИ** - Искусственный Интеллект

**СКО** - среднее квадратическое отклонение

## Методы машинного обучения

**DT** - Decision Tree, решающее дерево

**ET** - Extremely Randomized Trees, чрезвычайно случайный лес

**GNB** - Gaussian Naive Bayes, Гауссовский Наивный Байес

**kNN** - k Nearest Neighbours, k ближайших соседей

**LR** - Logistic Regression, логистическая регрессия

**MLP** - Multi - Layer Perceptron, многослойный персептрон

**NN** -Neural Network, нейронная сеть

**RF** - Random Forest, случайный лес

**RTE** - Random Trees Embedding, совершенно случайные деревья

**SVM** - Support Vector Machine, метод опорных векторов

**XGB, XGBoost** - Extreme Gradient Boosting, экстремальный градиентный бустинг

## Список литературы


1. Haleplidis, E. Software-Defined Networking (SDN): Layers and Architecture Terminology / E. Haleplidis, K. Pentikousis, S. Denazis, J.S. Salim, D. Meyer, O. Koufopavlou // RFC - 2015. – doi:10.17487/RFC7426
2. Росляков, А.В. Будущие сети: обзор подходов к новой телекоммуникационной парадигме / А.В. Росляков // Электросвязь. – 2020. - №9. - С. 30-37.
3. Росляков, А.В. Технологии реализации сетевых услуг NaaS / А.В. Росляков, М.В. Марыков // Под. ред. С.В. Бачевского; сост. А.Г. Владыко, Е.А. Аникевич/ АПИНО; IX Международная научно-техническая и научно-методическая конференция; сб. науч. ст. в 4 т. - СПб.: СПбГУТ. - 2020. - Т.1. – С. 707-711.
4. Росляков, А.В. Сетевое исчисление (Network Calculus) и его применение для оценки сетевых характеристик / А.В. Росляков, А.А. Лысиков; ПГУТИ. – Самара, 2019. — 222 с.
5. Silva, M.V.B. IDEAFIX: Identifying elephant flows in P4-based IXP networks / M.V.B. da Silva, A.S. Jacobs, R.J. Pfitscher, L.Z. Granville // 2018 IEEE Global Communications Conference (GLOBECOM), - 2018. - PP. 1-6.
6. Ghulam M.-U.D. Data learning and traffic classification by Machine Learning / M.-U.D. Ghulam, L.Z Qiang, Z. Jiangbin // Academicsera 18 th International Conference, - Sydney, Australia, 2018.
7. Gomes, R.L. A traffic classification agent for virtual networks based on QoS classes / R.L. Gomes, M.E.R. Madeira // IEEE Latin Am. Trans, - 2012. -10(3), - P. 1734–1741.
8. Troia, S. Machine-learning-assisted routing in SDN-based optical networks / S. Troia, N. Martin, A. Rodriguez, J.A. Hernandez // 44th European Conference on Optical Communication (ECOC), - At Rome, 2018. - URL: – doi:10.1109/ECOC.2018.8535437
9. Pham, Q.T. Deep Reinforcement Learning based QoS-aware Routing in Knowledge-defined networking / Q.T. Pham, Y. Hadjadj-Aoul, A. Outtagarts // 14th EAI International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, - Ho Chi Minh City, Vietnam, 2018. - PP.1-13.
10. Краснова, И.А. Классификация потоков трафика SDN-сетей методами машинного обучения в режиме реального времени / И.А. Краснова, В.А. Маньков // Труды международной научно-технической конференции «Информационные технологии и математическое моделирование систем 2019». – Одинцово, 2019. - С.65-68.- doi:10.36581/CITP.2019.31.51.016
11. Lin L. Framework for QoS-aware Traffic Classification Using Semi-supervised Machine Learning in SDNs / L. Lin, S.Wang, Luo, A. Min // 2016 IEEE International Conference on Services Computing (SCC), - 2016. – PP. 760-765, – doi:10.1109/SCC.2016
12. Singh, K. Near Real-time IP Traffic Classification Using Machine Learning / K. Singh, S. Agrawal, B. A Sohi, // International Journal of Intelligent Systems and Applications, - 2013. -№ 5. - PP. 83-93, - doi:10.5815/ijisa.2013.03.09
13. Iwai, T. Adaptive mobile application identification through in-network machine learning / T. Iwai, A. Nakao // 18th Asia-Pacific Network Operations and Management Symposium (APNOMS), - 2016. – PP. 1-6, - doi:10.1109/APNOMS.2016.7737226
14. Saqib, N. A. An effective empirical approach to VoIP traffic classification / N.A. Saqib, Y. Shakeel, M.A. Khan, H. Mehmood, M. Zia // Turkish Journal of Electrical Engineering & Computer Sciences,- 2017. - vol. 25, no. 2, - PP. 888-900.

15. Huang, N., Application identification system for SDN QoS based on machine learning and DNS responses / N. Huang, C. Li, C. Chen, I. Hsu// 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), - 2017. – PP. 407-410, - doi:10.1109/APNOMS.2017.8094160.
16. Huang, N. Application traffic classification at the early stage by characterizing application rounds / N. Huang, G. Jai, H. Chao, Y. Tzang, H. Chang // Inf. Sci., - 2013. - №232, - PP. 130-142, - doi: 10.1016/j.ins.2012.12.039.
17. Anantavrasilp, I. Automatic flow classification using machine learning / I. Anantavrasilp, T. Scholer // 15th International Conference on Software, Telecommunications and Computer Networks, - 2007. – PP. 1-6, - doi:10.1109/SOFTCOM.2007.4446129.
18. Dong, Y. Classification of Network Game Traffic Using Machine Learning. / Y. Dong, M. Zhang, R. Zhou /In: Yuan H., Geng J., Liu C., Bian F., Surapunt T. (eds) // Geo-Spatial Knowledge and Intelligence. GSKI 2017. Communications in Computer and Information Science, - Springer, Singapore, 2017. - vol 848, -doi:10.1007/978-981-13-0893-2_15
19. Chhabra, A. Classifying Elephant and Mice Flows, in High-Speed Scientific Networks / A. Chhabra, M. Kiran // 4th International Workshop on Innovating the Network for Data Intensive Science (INDIS), - 2017.
20. Zhang, C. Deep learning–based network application classification for SDN / C. Zhang, X. Wang, F. Li, Q. He, M. Huang // Transactions on Emerging Telecommunications Technologies, - 2017. -№ 29, -doi:10.1002/ett.3302
21. Lotfollahi, M. Deep Packet: A Novel Approach For Encrypted Traffic Classifcation Using Deep Learning / M. Lotfollahi, R.S.X. Zade, M.J. Siavoshani // Soft Comput 24(8): 1999, - 2017. - doi:10.1007/s00500-019-04030-2
22. Jamuna, A. Efficient Flow based Network Traffic Classification using Machine Learning /Jamuna A., S.E.E. Vinodh // International Journal of Engineering Research and Applications (IJERA), - 2013. - Vol. 3, Issue 2, - PP.1324-1328
23. Fan, Zh. Investigation of machine learning based network traffic classification / Zh. Fan, R. Liu // 14th International Symposium on Wireless Communication Systems, - 2017. - PP.1-6, - doi:10.1109/ISWCS.2017.8108090
24. Mestres, A. Knowledge-Defined Networking /A. Mestres, A. Rodríguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés, D. Meyer, S. Barkai, M.J. Hibbett, G. Estrada, K. Maruf, F. Coras, V. Ermagan, H. Latapie, C. Cassar, J. Evans, F. Maino, L.C. Walrand, A. Cabellos-Aparicio// Comput. Commun. Rev, - 2017. - ArXiv, abs/1606.06222
25. Parsaei, M.R. Network Traffic Classification using Machine Learning Techniques over Software Defined Networks / M.R. Parsaei, M.J. Sobouti, R. Javidan // International Journal of Advanced Computer Science and Applications (IJACSA), - 2017. - № 8(7), - doi:10.14569/IJACSA.2017.080729
26. Changhe, Y QoS-aware traffic classification architecture using machine learning and deep packet inspection in SDNs / Y. Changhe, J. Lan, J. Xie, Y. Hu // International congress of information and communication technology, - 2018. – PP. 1209–1216
27. Middleton, S.E. Scalable Classification of QoS for Real-Time Interactive Applications from IP Traffic Measurements / S.E. Middleton, S.Modafferi // Computer Networks, 107, - 2016. – doi:10.1016/j.comnet.2016.04.017
28. Pham, Q.T. Deep Reinforcement Learning based QoS-aware Routing in Knowledge-defined networking / Q.T. Pham, Y. Hadjadj-Aoul, A. Outtagarts// 14th EAI International Conference on

Heterogeneous Networking for Quality, Reliability, Security and Robustness, - 2018. - Ho Chi Minh City, Vietnam. - PP.1-13.
29. Alharbi, F. SDN-based mechanisms for provisioning quality of service to selected network flows / F. Alharbi// Theses and Dissertations,-Computer Science, 2018. – 72.
30. Aroussi, S. Survey on machine learning-based QoE-QoS correlation models / S. Aroussi, A. Mellouk // 2014 International Conference on Computing, Management and Telecommunications (ComManTel), - 2014. –PP. 200-204.
31. Alharbi, F. SDN-Survey on Machine Learning-based QoE-QoS Correlation Models: An adaptive machine learning-based QoE approach in SDN context for video-streaming services // F. Alharbi // Theses and Dissertations-Computer Science, - 2018. - 72.
32. Caicedo, O.M. Evalution de QoS usando tecnicas de machine learning / O.M. Caicedo // Universidad del Cauca, - 2019. – P. 19.
33. Letaifa, A.B. ML based QoE enhancement in SDN context: Video streaming case / A.B. Letaifa, G. Maher, S. Mouna // 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), - 2017. – PP.103-108.
34. Raumer, D. MonSamp: A distributed SDN application for QoS monitoring / D. Raumer, L. Schwaighofer, G. Carle // 2014 Federated Conference on Computer Science and Information Systems, - 2014. – PP. 961-968.
35. Huang, N. A dynamic QoS management system with flow classification platform for software-defined networks / N. Huang, I. Liao, H. Liu, S. Wu, C. Chou // 2015 8th International Conference on Ubi-Media Computing (UMEDIA),- 2015. – PP. 72-77.
36. Sapio, A. Scaling Distributed Machine Learning with In-Network Aggregation / A. Sapio, M. Canini, C. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D.R. Ports, P. Richtárik // Soft Computing manuscript, - 2019. -ArXiv, abs/1903.06701.
37. Бурлаков, М.Е. Исследование зависимости элементов двуклассификационной искусственной иммунной системы для обнаружения вторжений / М.Е. Бурлаков, М.Н. Осипов // Международная Конференция и молодёжная школа "Информационные технологии и нанотехнологии" (Конференция ИТНТ-2016).— 2016.—С.398-405.
38. Bakker, J.N. Traffic Classification with Machine Learning in a Live Network / J.N. Bakker, B. Ng, W.K. Seah, A. Pekár // 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), - 2019. – PP. 488-493.
39. Distributed Internet Traffic Generator, URL: http://www.grid.unina.it/software/ITG (дата обращения 21.08.2019).
40. Scapy. URL: https://scapy.net/ (дата обращения 01.02.2020).
41. CAIDA Data - Overview of Datasets, Monitors, and Reports. URL: https://www.caida.org/data/overview/ (дата обращения 01.02.2020).
42. SecRepo.com - Samples of Security Related Data URL: http://www.secrepo.com/ (дата обращения 01.02.2020).
43. ISCX Information Centre of Excellence for Tech Innovation URL: http://www.iscx.ca/datasets/ (дата обращения 01.02.2020).
44. Knowledge-Defined Networking/ URL: https://github.com/knowledgedefinednetworking/-knowledge-defined-networking (дата обращения 01.02.2020).
45. Tcpreplay - Pcap editiung and replaying utilities URL: https://tcpreplay.appneta.com/ (дата обращения 01.02.2020).

46. Netify DPI Engine - Open Source, URL: https://www.netify.ai/developer/netify-dpi-engine (дата обращения 01.02.2020).
47. Mockapetris P. Domain names - concepts and facilities. URL: https://tools.ietf.org/html/rfc1034 (дата обращения 01.02.2020).
48. tcptrace(1) - Linux man page. URL: https://linux.die.net/man/1/tcptrace (дата обращения 01.02.2020).
49. R&S®INTRA Intelligent network traffic analytics for communication service providers. URL: https://www.ipoque.com/products/analytics-solution-rsrintra. (дата обращения 01.02.2020).
50. OpenDPI. URL: https://github.com/ramanqul/OpenDPI (дата обращения 01.02.2020).
51. nDPI URL: https://github.com/ntop/nDPI (дата обращения 01.02.2020).
52. Application Layer Packet Classifier for Linux. URL: http://l7-filter.sourceforge.net/ (дата обращения 01.02.2020).
53. Libprotoident URL: https://github.com/wanduow/libprotoident (дата обращения 01.02.2020).
54. Network Based Application Recognition (NBAR). URL: https://www.cisco.com/c/en/us/products/ios-nx-os-software/network-based-application-recognition-nbar/index.html (дата обращения 01.02.2020).
55. Gringoli, F. GT: picking up the truth from the ground for internet traffic / F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, K.C. Claffy // Computer Communication Review, - 2009. - PP. 12-18.
56. Машинное обучение вместо DPI. Строим классификатор трафика URL: https://habr.com/ru/post/304926/ (дата обращения 01.02.2020).
57. Ванюшина А.В. Классификация IP-трафика в компьютерной сети с использованием алгоритмов машинного обучения: дис. ... канд. техн. наук:05.13.15/ Ванюшина Анна Вячеславовна М. 2019. - 205 с.
58. Manpage of TCPDUMP, URL: https://www.tcpdump.org/manpages/tcpdump.1.html (дата обращения 01.02.2020).
59. Libpcap Library, URL: https://www.sciencedirect.com/topics/computer-science/libpcap-library (дата обращения 01.02.2020).
60. NetFlow, URL: http://xgu.ru/wiki/NetFlow (дата обращения 01.02.2020).
61. Harrington, D. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks / D. Harrington, R. Presuhn, B. Wijnen // RFC, - 2002, - doi:10.17487/RFC3411
62. Pasquini, R. Learning end-to-end application QoS from openflow switch statistics / R. Pasquini, R. Stadler // 2017 IEEE Conference on Network Softwarization (NetSoft), - 2017. – PP. 1-9.
63. Yalda, K.G. Getting traffic statistics from network devices in an SDN environment using OpenFlow / K.G. Yalda, I.T. Okumus // Proceedings of 2015 Information Technology and Systems (ITaS), - 2015. - Olympic Village, Sochi, Russia, - PP. 951–956.
64. Краснова, И.А. Алгоритм динамической классификации потоков в мультисервисной SDN-сети / И.А. Краснова, В.А. Маньков // T-Comm: Телекоммуникации и транспорт. - 2017. – Т.11, №12. - С. 37-42.
65. Краснова, И.А. Задача управления трафиком с динамическим определением QoS в мультисервисных SDN сетях / И.А. Краснова, В.А. Маньков // Сборник трудов XI международной отраслевой научно-технической конференции «Технологии информационного общества» / МТУСИ. Москва, 2017. - С.67-68.

66. Chowdhury, S.R. PayLess: A low cost network monitoring framework for Software Defined Networks / S.R. Chowdhury, M.F. Bari, R. Ahmed, R. Boutaba // 2014 IEEE Network Operations and Management Symposium (NOMS), - 2014. – PP. 1-9.
67. Bosshart, P. P4: programming protocol-independent packet processors / P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker // Computer Communication Review, - 2014. - №44, - PP. 87-95.
68. Liu, Z. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon / Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, V. Braverman // SIGCOMM '16, - 2016.
69. Nathan, V. Demonstration of the Marple System for Network Performance Monitoring / V. Nathan, S. Narayana, A. Sivaraman, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, C. Kim // SIGCOMM Posters and Demos '17, - 2017.
70. Krasnova, I.A. Collection of Individual Packet Statistical Information in a Flow Based on P4-switch. / I.A. Krasnova, V.Yu. Deart, V.A. Mankov // In: Hu Z., Petoukhov S., He M. (eds) Advances in Intelligent Systems, Computer Science and Digital Economics. CSDEIS 2019. / Advances in Intelligent Systems and Computing; Springer, Cham - 2020. - vol 1127, - pp.106-117, doi:10.1007/978-3-030-39216-1_11
71. Машинное обучение для людей: Разбираемся простыми словами. URL: https://vas3k.ru/blog/machine_learning/ (дата обращения: 01.02.2020).
72. Медведева, Е.Г. Применение кластеризации в задачах размещения подвижных точек доступа в воздушно-наземных беспроводных сетях / Е.Г. Медведева, Э.М. Хайров, Н.А. Поляков, Ю.В. Гайдамака // Системы и средства информатики. - 2020. - Т.30, № 4. - С. 25-37.
73. Ponomareva, L.A. Neural network information management model for a large educational complex / L.A. Ponomareva, O.N. Romashkova, Y.V. Gaidamaka // CEUR Workshop Proceedings. – 2020. –vol. 2639. - pp. 5-21.
74. Рыжков Д.О. Определение протокола прикладного уровня для анализа сетевого трафика с применением алгоритмов машинного обучения / Рыжков Д.О. // Материалы IX Международной студенческой научной конференции «Студенческий научный форум», -2017.
75. Weka 3 - Data Mining with Open Source Machine Learning Software in Java. URL: https://www.cs.waikato.ac.nz/ml/weka/ 2017032799 (дата обращения: 01.02.2020).
76. RapidMiner Studio. URL: https://rapidminer.com/get-started/ (дата обращения: 01.02.2020).
77. Scikit-learn: machine learning in Python. URL: https://scikit-learn.org/stable/ (дата обращения: 01.02.2020 ).
78. STATGRAPHICS URL: http://www.statgraphics.com/ (дата обращения: 01.02.2020).
79. STATISTICA URL: http://statsoft.ru/products/ (дата обращения: 01.02.2020).
80. MATLAB URL: https://exponenta.ru/products/matlab (дата обращения: 01.02.2020).
81. Кросс-валидация (Cross-validation). URL: https://long-short.pro/post/kross-validatsiya-cross-validation-304/ (дата обращения: 01.02.2020).
82. Google Scholar. URL: https://scholar.google.com/ (дата обращения: 04.01.2021).
83. Latah, M. Artificial Intelligence Enabled Software Defined Networking: A Comprehensive Overview / M. Latah, L. Toker // IET Networks 8, - 2018. – doi:10.1049/iet-net.2018.5082
84. Buczak, A.L. A survey of data mining and machine learning methods for cyber security intrusion detection / A.L. Buczak, E. Guven // IEEE Commun. Surveys Tutorials, - 2016. -vol. 18, no. 2, - PP. 1153–1176

85. Hodo, E. Shallow and Deep Networks Intrusion Detection System: A Taxonomy and Survey / E. Hodo, X.J. Bellekens, A.W. Hamilton, C. Tachtatzis, R.C. Atkinson // ArXiv, abs/1701.02145, - 2017.
86. Гетьман, А.И. Обзор задач и методов их решения в области классификации сетевого трафика / А.И. Гетьман, Ю.В. Маркин, Е.Ф. Евстропов, Д.О. Обыденков // Труды ИСП РАН, - Т.29 (3). – 2017. – С. 117-150. - doi:10.15514/ISPRAS-2017-29(3)-8
87. Краснова, И.А. Виртуализация сетевых функций и программно-конфигурируемые сети: учебное пособие / И.А. Краснова, В.А. Маньков, А.Е. Панов; МТУСИ – Москва, 2020.– 126 с.
88. Краснова, И.А. Анализ перспективных подходов и исследований по классификации потоков трафика для поддержания QoS методами ML в SDN-сетях / И.А. Краснова, В.Ю. Деарт, В.А. Маньков // Вестник СибГУТИ - 2021. - №1. - С. 3-22.
89. Packet traces from WIDE backbone URL: http://mawi.wide.ad.jp/mawi/ (дата обращения: 01.12.2019).
90. Cho, K. Traffic Data Repository at the WIDE Project / K. Cho, K. Mitsuya, A. Kato// USENIX Annual Technical Conference, FREENIX Track, -2000.
91. Habibi, L.A. Characterization of Tor Traffic using Time based Features / L.A. Habibi, G.G. Draper, M. Mamun, A. Ghorbani // Proceedings of the 3rd International Conference on Information Systems Security and Privacy – 2017. – Vol. 1, - PP. 253-262, - doi:10.5220/0006105602530262
92. Moore, A. Discriminators for Use in Flow-Based Classification / A. Moore, D. Zuev, M. Crogan, // Technical report, Queen Mary, University of London, - 2005.
93. Шелухин, О.И. Классификация IP-трафика методами машинного обучения / О.И. Шелухин, С.Д. Ерохин, А.В. Ванюшина /Под ред. О.И. Шелухина; М.: Горячая линия – Телеком, - 2018. - 282 с: ил.
94. Krasnova, I.A. Development of a Feature Matrix for Classifying Network Traffic in SDN in Real-Time Based on Machine Learning Algorithms / I.A. Krasnova, V.Yu. Deart, V.A. Mankov // 2020 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC) - Moscow, 2020. - pp. 1-9. - doi:10.1109/MoNeTeC49726.2020.9258314
95. Buitinck, L. API design for machine learning software: experiences from the scikit-learn project / L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, G. Varoquaux // ArXiv, abs/1309.0238, - 2013.
96. Sheluhin, O.I. Influence of training sample structure on traffic application efficiency classificationusing machine-learning methods / O.I. Sheluhin, A.G. Simonyan, A.V. Vanyushina // T-Comm, - 2017. - vol. 11, no.2, - PP. 25-31. (inRussian)
97. Perera, P. A Comparison of Supervised Machine Learning Algorithms for Classification of Communications Network Traffic /P. Perera, Y.C. Tian, C. Fidge, W. Kelly // Liu D., Xie S., Li Y., Zhao D., El-Alfy ES. (eds); Neural Information Processing. ICONIP 2017 // Lecture Notes in Computer Science, - vol 10634, Springer, Cham, 2017. – doi:10.1007/978-3-319-70087-8_47
98. Erman, J. QRP05-4: Internet Traffic Identification using Machine Learning / J. Erman, A. Mahanti, M. Arlitt // IEEE Globecom, - 2006. – PP. 1-6.
99. Samuylov, A. Characterizing Resource Allocation Trade-Offs in 5G NR Serving Multicast and Unicast Traffic / A. Samuylov, D. Moltchanov, R. Kovalchukov, R. Pirmagomedov, Y. Gaidamaka, S. Andreev, Y. Koucheryavy, K. Samouylov // IEEE Transactions on Wireless Communications. – 2020. – vol. 19(5). - pp. 3421-3434. – doi: 10.1109/TWC.2020.2973375

100. Breiman, L. Classification and Regression Trees / L. Breiman, J. Friedman, R. Olshen, C. Stone // Wadsworth, Belmont, CA, - 1984.
101. Pedregosa, F. Scikit-learn: Machine Learning in Python / F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, G. Louppe, P. Prettenhofer, R. Weiss, R.J. Weiss, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay // J. Mach. Learn. Res., - 12, -2011. - PP. 2825-2830.
102. Hastie, T. The Elements of Statistical Learning / T. Hastie, R. Tibshirani, J. Friedman // The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition - Springer, New York, NY, - 2009. – doi:10.1007/978-0-387-84858-7
103. Breiman, L. Random Forests / L. Breiman // Machine Learning - 45(1), - 2001.- PP. 5-32.
104. Chen, T. XGBoost: A Scalable Tree Boosting System / T. Chen, C. Guestrin // Conference: the 22nd ACM SIGKDD International Conference, - 2016. – doi:10.1145/2939672.2939785
105. Introduction to Boosted Trees // URL: https://xgboost.readthedocs.io/en/latest/tutorials/model.html (дата обращения 03.03.2020)
106. McCallum, A. A Comparison of Event Models for Naive Bayes Text Classification / A McCallum, K. Nigam // AAAI/ICML-98 Workshop on Learning for Text Categorization, Technical Report WS-98-05. AAAI Press, - 1998. - PP. 41–48.
107. Tolles, J. Logistic Regression Relating Patient Characteristics to Outcomes / J. Tolles, W. Meurer // JAMA, - 316 (5): 533–4, - 2016. – doi:10.1001/jama.2016.7653
108. Goldberger, J. Neighbourhood Components Analysis / J. Goldberger, S. Roweis, G. Hinton, R. Salakhutdinov // Advances in Neural Information Processing Systems,- Vol. 17, - 2005. - PP. 513-520.
109. Hinton, G.E. Connectionist learning procedures / G.E. Hinton // Artificial intelligence 40.1, - 1989. – PP. 185-234.
110. Glorot, X. Understanding the difficulty of training deep feedforward neural networks / X. Glorot, Y. Bengio // International Conference on Artificial Intelligence and Statistics, - 2010.
111. Krasnova, I.A. Evaluation of the Effect of Preprocessing Data on Network Traffic Classifier Based on ML Methods for Qos Predication in Real-Time / I.A. Krasnova, V.Yu. Deart, V.A. Mankov // In: Hu Z., Petoukhov S., He M. (eds) Advances in Artificial Systems for Medicine and Education IV. AIMEE 2020. / Advances in Intelligent Systems and Computing, Springer, Cham. -2021. - vol 1315, - pp. 55-64. - doi:10.1007/978-3-030-67133-4_5
112. Alothman, B. Raw Network Traffic Data Preprocessing and Preparation for Automatic Analysis / B. Alothman // 2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), - 2019. – PP. 1-5. – doi:10.1109/CyberSecPODS.2019.8885333
113. Kahya-Özyirmidokuz, E. Characterization of Network Traffic Data:A Data Preprocessing and Data Mining Application / E. Kahya-Özyirmidokuz, A. Gezer, C. Çiflikli// IARIA, 2012 BarcelonaVolume: 978-1-61208-242-4, - 2012.
114. Kumar, A. D. The Effect of Normalization on Intrusion Detection Classifiers (Naïve Bayes and J48) / A.D. Kumar, S.R. Venugopalan // International Journal on Future Revolution in Computer Science & Communication Engineering, - 2017. - № 3, - PP. 60-64.
115. Wang, W. Attribute Normalization in Network Intrusion Detection / W. Wang, X. Zhang, S. Gombault, S.J. Knapskog // 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks, - 2009. – PP. 448-453.
116. Hotelling, H. Analysis of a complex of statistical variables into principal components / H. Hotelling // Journal of Educational Psychology, - 1933. - № 24, - PP. 417–441, 498–520.

117. Smola, A.J. A Tutorial on Support Vector Regression / A.J. Smola, B. Schölkopf // Statistics and Computing archive, - vol. 14 (3), - 2004. - PP. 199-222.
118. Preprocessing data, URL: https://scikit-learn.org/stable/modules/preprocessing.html (дата обращения 15.12.2019).
119. Yeo, I.-K. A new family of power transformations to improve normality or symmetry / I.-K. Yeo // Biometrika, 87(4), - 2000. – PP. 954–959. – doi:10.1093/biomet/87.4.954
120. Шелухин, О.И. Классификация зашифрованного трафика мобильных приложений методом машинного обучения/ О.И. Шелухин, В.В. Барков, М.В. Полковников //Вопросы кибербезопасности, - №4 (28), -2018. – doi:10.21681/2311-3456-2018-4-21-28
121. Wang, C. Network Traffic Classification with Improved Random Forest /C. Wang, T. Xu X. Qin // 2015 11th International Conference on Computational Intelligence and Security (CIS), - 2015.
122. Zhai, Y. Random Forest based Traffic Classification Method In SDN / Y. Zhai, X. Zheng // 2018 International Conference on Cloud Computing, Big Data and Blockchain, -2018.- - PP. 1-5.
123. Краснова, И.А. Анализ влияния параметров алгоритмов Machine Learning на результаты классификации трафика в режиме реального времени / И.А. Краснова // T-Comm: Телекоммуникации и транспорт. - 2021. – Т.15, №9. - С. 24-35.- doi: 10.36724/2072-8735-2021-15-9-24-35
124. Takyi, K. Clustering Techniques for Traffic Classification: A Comprehensive Review / K. Takyi, A. Bagga, P. Goopta // 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), - 2018. – PP. 224-230.
125. Ерохин, С.Д. Использование алгоритма кластеризации kMeans для идентификации сетевого трафика / С.Д. Ерохин, А.В. Ванюшина // Электросвязь, - №12, - 2018. - С.48-49.
126. Mahmood, A. Hierarchical summarization techniques for network traffic / A. Mahmood, C. Leckie, R. Islam, Z. Tari // 2011 6th IEEE Conference on Industrial Electronics and Applications, - 2011. – PP. 2474-2479.
127. Mahaling, G.S. Network Traffic Analysis of Hierarchical Data Using Clustering / G.S. Mahaling // International Journal of Science and Research (IJSR), - Vol. 6 (3), - 2017. – PP. 1996-1999.
128. Wang, Y. Network traffic clustering using Random Forest proximities / Y. Wang, Y. Xiang, J. Zhang // 2013 IEEE International Conference on Communications (ICC), - 2013. – PP. 2058-2062. - doi:10.1109/ICC.2013.6654829
129. Krasnova, I.A. Agglomerative Clustering of Network Traffic Based on Various Approaches to Determining the Distance Matrix / I.A. Krasnova, V.Yu. Deart, V.A. Mankov // 2021 28th Conference of Open Innovations Association (FRUCT) - Moscow, 2021. - pp. 81-88. - doi:10.23919/FRUCT50888.2021.9347616
130. Zhang, W. Graph Degree Linkage: Agglomerative Clustering on a Directed Graph / W. Zhang, X. Wang, D. Zhao, X. Tang, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, C. Schmid // Computer Vision – ECCV 2012. // Lecture Notes in Computer Science. Springer Berlin Heidelberg, - 2012. - № 7572, - PP. 428–441.
131. Lance, G.N. A General Theory of Classificatory Sorting Strategies: Hierarchical Systems / G.N. Lance, W.T. Williams// The Computer Journal, - Vol. 9 (4), - 1967. – PP. 373–380.
132. Louppe, G. Ensembles on Random Patches / G. Louppe, G. Geurts// Machine Learning and Knowledge Discovery in Databases, - 2012. – PP. 346-361.
133. Geurts, P. Extremely randomized trees / P. Geurts, D. Ernst, L. Wehenkel // Mach Learn, - № 63, - 2006. - PP. 3–42. – doi:10.1007/s10994-006-6226-1

134. Ho, T. The random subspace method for constructing decision forests / T. Ho // Pattern Analysis and Machine Intelligence, - № 20(8) - 1998. - PP. 832-844.
135. One-Hot Encoding in Python – Implementation using Sklearn, URL: https://www.journaldev.com/45203/one-hot-encoding-in-python (дата обращения 01.01.2020).
136. Vinh, N. X. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? / N.X. Vinh, J. Epps, J. Bailey, // ICML '09, - 2009.
137. Nguyen, X.V. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance / X.V. Nguyen, J. Epps, J. Bailey // J. Mach. Learn. Res. 11, - 2010. – PP.2837-2854.
138. Rand, W. M. Objective criteria for the evaluation of clustering methods / W.M. Rand // Journal of the American Statistical Association, № 66(336), - 1971. - PP.846–850.
139. He, Z. k-anmi: A mutual information based clustering algorithm for categoricaldata / Z. He, X. Xu, S. Deng // Inf. Fusion, - 2008. - № 9(2), PP. 223–233.
140. Maaten, L.V.D. Visualizing Data using t-SNE / L.V.D. Maaten, E.H. Geoffrey // Journal of Machine Learning Research, - № 9, -2008. - PP. 2579-2605.
141. Maaten, L.V.D. Accelerating t-SNE using Tree-Based Algorithms / L.V.D. Maaten // Journal of Machine Learning Research 15(Oct), - 2014. – PP.3221-3245.
142. Krasnova, I.A. An Extensible Network Traffic Classifier Based on Machine Learning Methods / I.A. Krasnova, V.Yu. Deart, V.A. Mankov // In: Hu Z., Petoukhov S., He M. (eds) Advances in Intelligent Systems, Computer Science and Digital Economics II. CSDEIS 2020. / Advances in Intelligent Systems and Computing, Springer, Cham. -2021. – vol. 1402, - PP. 10-19. – doi: 10.1007/978-3-030-80478-7_2
143. OpenFlow Controller Benchmarking Methodologies // Open Networking Foundation 2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303, - 2016.
144. Смелянский, Р. Технологии реализации программно конфигурируемых сетей: Overlay vs OpenFlow / Р. Смелянский // Журнал сетевых решений, -Т. 20, №4 – 2014. -С.53-55.
145. Росляков, А.В. Сеть 2030 – взгляд МСЭ-Т на будущее сетей фиксированной связи / А.В. Росляков // Первая миля. – 2021. - №4. - С. 50-59.
146. Samuylov, A. Characterizing Resource Allocation Trade-Offs in 5G NR Serving Multicast and Unicast Traffic / A. Samuylov, D. Moltchanov, R. Kovalchukov, R. Pirmagomedov, Y. Gaidamaka, S. Andreev, Y. Koucheryavy, K. Samouylov // IEEE Transactions on Wireless Communications. – 2020. – vol. 19(5). - pp. 3421-3434. – doi: 10.1109/TWC.2020.2973375
147. Власкина, А.С. Анализ вероятностно-временных характеристик обслуживания эластичного трафика с минимальной скоростью в сегменте беспроводной сети с нарезкой радиоресурсов / А.С. Власкина, Н.А. Поляков, И.А. Гудкова, Ю.В. Гайдамака // Известия Саратовского университета. Новая серия. Серия: Математика. Механика. Информатика. - 2020. - Т.20, № 3 - С. 378-387. - doi: 10.18500/1816-9791-2020-20-3-378-387
148. $P4_{16}$ Language Specification, ver. 1.2.1 // The P4 Language Consortium 2020-06-11, - 2020.
149. P4 Language and Related Specifications. URL: https://p4.org/specs/ (дата обращения: 11.07.2020).
150. Mininet. URL: http://mininet.org/ (дата обращения: 05.07.2018).
151. Open vSwitch. URL: https://www.openvswitch.org/ (дата обращения: 05.07.2018).
152. Ethernet — DIX V2. URL: https://www.ibm.com/docs/en/zos/2.2.0?topic=internetworking-ethernet-dix-v2# (дата обращения 01.02.2020).
153. Internet Protocol // darpa internet program protocol specification, RFC 791, - 1981.

154. Transmission Control Protocol // darpa internet program protocol specification, RFC 793, - 1981.
155. Postel, J. User Datagram Protocol / J. Postel // darpa internet program protocol specification, RFC 768, - 1980.
156. Emma, D. Analysis and experimentation of an open distributed platform for synthetic traffic generation / D. Emma, A. Pescapé, G. Ventre // 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004),- 2004. - PP. 277-283.
157. Avallone, S. Distributed Internet Traffic Generator (D-ITG): analysis and experimentation over heterogeneous networks / S. Avallone, A. Pescapé, G. Ventre // Poster at International Conference on Network Protocols, ICNP 2003 November 2003, - Atlanta - Georgia (USA), 2003.
158. Recommendation ITU-T G.711.1 (09/2012) Wideband embedded extension for ITU-T G.711 pulse code modulation. – 2012.
159. Рекомендация G.729.1 (05/2006) Встроенный кодер G.729 с переменной скоростью передачи: двоичный поток широкополосного масштабируемого кодера со скоростями 8-32 кбит/с, способный взаимодействовать с G.729. – 2006.
160. Sample Captures - The Wireshark Wiki. URL: https://wiki.wireshark.org/SampleCaptures (дата обращения: 01.02.2020).
161. Tstat - TCP STatistic and Analysis Tool. URL: http://tstat.polito.it/traces-skype.shtml (дата обращения: 01.02.2020).
162. Table 1. – Name Mappings of categories. URL: https://goo.gl/Wwnivx (дата обращения: 01.02.2020).
163. Dupay, A. Netmate: A network management environment / A. Dupay, S. Sengupta, O. Wolfson, Y. Yemini // IEEE Network 5, - 1991. –PP. 35-40. – doi:10.1109/65.75840
164. Asadollahi, S. Experimenting with scalability of floodlight controller in software defined networks / S. Asadollahi, B. Goswami // Conference: 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), - 2017. – doi:10.1109/ICEECCOT.2017.8284684

# Приложение А. Сравнительная характеристика основных работ по классификации потоков трафика для обеспечения QoS методами машинного обучения в SDN-сетях

Таблица П.1. Часть 1: Формирование базы данных

| № | Статья | Формирование базы данных | | | |
|---|---|---|---|---|---|
| | | Способ получения базы данных | Состав базы данных | Мониторинг трафика | Маркировка образцов |
| 1 | [10] | Генерирование трафика с помощью *D-ITG* | 1000 потоков. Онлайн-игры: *Quake3, CSa, CSi,* голосовой трафик: кодеки *G.711.1* и *G.729.2* | Собственный метод[70] | *DSCP*-поле средствами генератора *D-ITG* |
| 2 | [11] | Использование базы данных реального трафика исследовательской группы *Broadband Communication Research Group in UPC* | 3377 потоков: Голос/видео *Conference Skype, QQ, Google Hangout*, интерактивные данные *Web, Http Services*, потоковое видео *PPStream, Vimeo, SopCast, Putlocker*, Передача данных *Transfer FTP, Torrent, Dropbox* | Не разработан | Модуль *DPI* |
| 3 | [12] | Реальный трафик, собранный на индивидуальном ПК | 2000 потоков; 8 интернет-приложений: *WWW*, электронная почта, веб-медиа, *P2P*, данные *FTP*, мгновенные сообщения, *VoIP* и обновления программного обеспечения (*SU*) | *Wireshark* | - |
| 4 | [7] | Реальный трафик, собранный на индивидуальном ПК | *Skype, Gtalk, MSN, SNMP, IGMP, HTTP, HTTPS, DNS, FTP, Telnet, P2P, SMTP, YouTube* | *libpcap++* | - |
| 5 | [13] | Реальный трафик, собранный с помощью смартфонов | Мобильные приложения | - | Специальное приложение на *Python*, помечающее пакеты до отправки по используемым процессам |
| 6 | [14] | Открытые базы данных, выложенные в интернете [160, | *Skype, Yahoo Messenger, Gtalk, MSN Messenger, Eyebeam-Asterisk-Blink, ooVoo, Zfone-Asterisk-Xlite,* | Не разработан | Маркировка, предложенная в готовых |

| № | Статья | Формирование базы данных | | | |
|---|---|---|---|---|---|
| | | Способ получения базы данных | Состав базы данных | Мониторинг трафика | Маркировка образцов |
| | | 161] | *Tencent QQ, Trillian IM, TEAMtalk, Mixed* (голосовой и неголосовой трафик), *Facebook, Email server (Hotmail, Yahoo Mail, Gmail), Online*-игры, *Online live TV (Geo, Duniya), Torrent, YouTub*e | | базах данных |
| 7 | [15] | Генерация трафика с помощью приложений на разных платформах: *Android 4.4.4, iOS 8.0.2, Windows 7, Mac OS X, Web* | 6525 потоков, 294 приложения [162] | - | Маркировка пакетов до их отправки, основанная на запущенных процессах |
| 8 | [16] | Реальный трафик в сети университета и имитация работы пользовательских приложений | По 100 потоков различных приложений | Не разработан | Разметка через *MSN* и *DNS* |
| 9 | [17] | Реальный трафик, собранный на 4-х индивидуальных ПК | *first-person shooters, Telnet, SSH, media stream, web, email* | *Flow Stat* | Физическая изоляция: на каждом компьютере используется свой класс трафика |
| 10 | [18] | Реальный трафик, собранный на индивидуальном ПК | 60 потоков данных, по 30 мин длительность: *Fantasy Westward Journey, against the war, furnace stone legend,LOL, DOTA2* | *Wireshark* | - |
| 11 | [19] | Реальный трафик сайта *Esnet* | Данные, собранные с 3-х маршрутизаторов в теч 1 мес | *Netflow* | - |

| № | Статья | Формирование базы данных | | | |
|---|---|---|---|---|---|
| | | Способ получения базы данных | Состав базы данных | Мониторинг трафика | Маркировка образцов |
| 12 | [6] | Реальный трафик, полученный на производственной сети предприятия, на которую установили *HP VAN SDN*-контроллер и коммутатор *HP E3800* с поддержкой *OpenFlow*. Трафик с одного хоста | 500 потоков для каждого из приложений: *YouTube, Vimeo, Facebook, LinkedIn, Skype, Bit Torrent, веб-браузер (HTTP)* и *Dropbox* | Собственный метод, разработанный для сетей *OpenFlow* | Физическая изоляция на порту и маркировка с помощью процессов запускаемых приложений |
| 13 | [20] | Открытый набор данных Мура из университета Кембридж | 10 наборов данных, записанных в разные дни | *tcptrace* | *tcptrace* |
| 14 | [21] | Отрытый набор данных *ISCX VPN-nonVPN* [43] | Приложения *AIM chat 5K, Email 28K, Facebook 2502K, FTPS 7872K, Gmail 12K, Hangouts 3766K, ICQ 7K, Netix 9K, SCP 448K, SFTP 418K, Skype 2872K, Spotify 40K, Torrent 70K, Tor 202K, Voipbuster 842K, Vimeo 146K, YouTube 251K* | Не разработан | Маркировка, предложенная в готовых базах данных |
| 15 | [22] | Реальный трафик, захваченный с помощью *Tcpdump* в сети Университета Каруни | 3500 потоков | *Tcpdump* и *Netmate* [163] | - |
| 16 | [23] | Открытый набор данных Мура из университета Кембридж | 10 наборов данных, записанных в разные дни | *tcptrace* | *tcptrace* |
| 17 | [24] | Реальный трафик, воспроизведенный на лабораторной установке с помощью *tcpreply* | 150+200 потоков данных | - | Приложения |
| 18 | [25] | Трафик, полученный на тестовом стенде, с двумя хостами, контроллером *Floodlight* [164] и одним *OVS* | 65000 потоков: *FTP, HTTP, instant messaging*, потоковое видео и *P2P* | средствами *Openflow* 20 раз за 1 с | Заполняется вручную |

| № | Статья | Формирование базы данных | | | |
|---|---|---|---|---|---|
| | | Способ получения базы данных | Состав базы данных | Мониторинг трафика | Маркировка образцов |
| 19 | [26] | Реальный трафик, собранный на индивидуальном ПК | 637690 потоков | - | Модуль *DPI* |
| 20 | [27] | Реальный трафик, собранный на индивидуальном ПК | *Minecraft, VLC, Quake 3, VoIP* | *tcpdump* | *tcpdump* и ручная маркировка |

Таблица П.2. Часть 2: Формирование матрицы признаков

| № | Статья | Формирование матрицы признаков | | |
|---|---|---|---|---|
| | | Шаблоны трафика | Признаки | Классы |
| 1 | [10] | Односторонний *UDP*-поток, идентифицируемый по *5-Tuple* | По каждому из первых 10 пакетов: длина пакета, время между приходом каждого из двух пакетов из одного потока, мин, макс, сред и сумм параметры; скорость поступления пакетов | Тип приложения: *Quake3, CSa, CSi*, голосовой трафик: кодеки *G.711.1* и *G.729.2* |
| 2 | [11] | Двухсторонние *UDP/TCP* потоки | Параметр Херста и среднее значение длины пакетов поступающих в одно и в другое направление, порт источника и назначения, медиана длины пакетов от источника к назначению, минимальная длина пакета от назначения к источнику, количество пакетов и т.д. | Категории: Голосовой трафик (*GoogleVoice*), видео-конференции (*Skype, GoogleTalk*), потоковое видео (*USstream, Sopcast*), передача данных (*FTP, Mega*), интерактивные данные (*SSH, Telnet*), категория по умолчанию (*Best effort*) |
| 3 | [12] | - | Минимальное, максимальное, среднее значение, дисперсия и общее число пакетов, среднее число пакетов в секунду, размер пакета, продолжительность | Категории, собранные в 8 интернет – приложений |
| 4 | [7] | - | - | Категории: аудио, управление, видео, данные |

| № | Статья | Формирование матрицы признаков | | |
|---|---|---|---|---|
| | | Шаблоны трафика | Признаки | Классы |
| 5 | [13] | Двухсторонние *TCP*-потоки | IP назначения, порт назначения, порт источника, СКО, дисперсия, максимум и медиана длины пакетов от источника к назначению и в обратном направлении, размеры окон в пакетах *SYN*, отношение количества *ACK-PSH* к общему кол-ву *TCP*-флагов, общее число пакетов | Тип приложения: мобильные приложения |
| 6 | [14] | Однонаправленные *TCP/UDP*-потоки | Направление, протокол, количество пакетов, длительность потока, среднее значение, СКО размера пакетов, скорость передачи пакетов, минимальный и максимальный размер пакетов | Категории трафика: голосовой/не голосовой трафик |
| 7 | [15] | Двухсторонние *UDP/TCP*-потоки | 142 признака: количество пакетов, размеры пакетов, время передачи, время между прибытием пакетов, направление передачи, пропускная способность, размеры окон и статистические данные (итого, минимум, максимум, медиана, среднее, дисперсия и соотношение обоих направлений) и т.д. | Идентификация приложений: *coursera, Facebook, Gmail, Instagram* и т.д. |
| 8 | [16] | Раунды *MSN*, *TCP* и *HTTP* | Размер передаваемого прикладного уровня, пропускная способность, время прибытия и время отклика на разных раундах установления соединения (протокол *MSN*), номер порта сервера, тип протокола транспортного уровня (*TCP* или *UDP*), флаг первого отправителя данных (клиент или сервер) и информация о размере. | Категории: потоковые данные (*ftp* и *xunlei*), *Web* (*http* и *https*), почта (*smtp* и *pop3*), *P2P* (*eDonkey, Bittorrent, SKYPE, Fasttrack, gnutella, kugoo, soulseek* и *peerenabler*), сервисы (*DNS*), мессенджеры (*aim, yahoo, qq, msnmessenger*, и *jabber*), интерактив (*telnet* и *irc*), игры (*doom3* и *xboxlive*), другое (*SSH* и др) |
| 9 | [17] | Двухсторонние *UDP/TCP*-потоки | Тип протокола, порт назначения, время сессии, количество пакетов: всего, отправлено, получено, отношение кол-ва отправленных к кол-ву полученных, пропускная способность: пиковая, средняя, отклонения | 4 Категории: трафик реального времени, чувствительный к задержкам (*first-person shooters*), трафик, реального времени, терпимый к задержкам (*Telnet, SSH*), потоковое видео (*media stream*), интерактивный класс (*web, email*) |

| № | Статья | Формирование матрицы признаков | | |
|---|---|---|---|---|
| | | Шаблоны трафика | Признаки | Классы |
| 10 | [18] | Двухсторонние *UDP/TCP*-потоки | Отношение количества полученных и переданных данных, энтропия информации о размере пакетов переданных и скорость передачи пакетов | Типы приложений (игр): *Fantasy Westward Journey, against the war, furnace stone legend,LOL, DOTA2* |
| 11 | [19] | Однонаправленные *TCP/UDP*-потоки | Размер (в байтах) передачи данных и длительность потока | Характер трафика: *Elephant* и *Mice* |
| 12 | [6] | Однонаправленные *TCP/UDP*-потоки | Длительность потока, количество переданных пакетов и байт | Тип приложения |
| 13 | [20] | Однонаправленные *TCP/UDP*-потоки | Длительность потока, время поступления пакета и количество пакетов в потоке | Категории: загрузка данных (*ftp*, базы данных *postgress, sqlnet oracle, ingress*), интерактивный трафик (*SSH, klogin, rlogin, telnet*), почта (*imap, pop2/3, smtp*, сервисы *X11, dns, ident, ldap, ntp*), веб (*www, P2P: KaZaA, BitTorrent, GnuTella),* интернет-атаки (вирусы), игры (*Microsoft Direct Play*), мультимедиа (*Windows Media Player, Rea*) |
| 14 | [21] | Однонаправленные *TCP/UDP*-потоки | нет | Категории: Чат, *Email*, передача данных, потоковые данные, *Torrent, VoIP,VPN:* Чат, *VPN: Email, VPN:* передача данных, *VPN:* потоковые данные, *VPN:Torrent, VPN:VoIP* |
| 15 | [22] | Двухсторонние *UDP/TCP*-потоки | Количество пакетов, размер пакета, время между поступлениями, длительность потока, Процент *IP*-пакетов с определенными размерами и процентами потоков с определенными размерами и длительностью пакетов, порты уровня транспортного протокола, размеры окон *TCP* | Категории: *WWW, E-MAIL, CHAT, P2P, FTP, IM (Instant Message), VoIP* |

| № | Статья | Формирование матрицы признаков | | |
|---|---|---|---|---|
| | | Шаблоны трафика | Признаки | Классы |
| 16 | [23] | Двухсторонние *UDP/TCP*-потоки | Порт сервера, минимальный и максимальный размер *TCP*-сегмента, отправленного от клиента к серверу, размер окна, количество пакетов с флагом *PUSH* | Категории: загрузка данных (*ftp*, базы данных *postgress, sqlnet oracle, ingress*), интерактивный трафик (*SSH, klogin, rlogin, telnet*), почта (*imap, pop2/3, smtp*, сервисы *X11, dns, ident, ldap, ntp*), веб (*www, P2P: KaZaA, BitTorrent, GnuTella),* интернет-атаки (вирусы), игры (*Microsoft Direct Play*), мультимедиа (*Windows Media Player, Rea*) |
| 17 | [24] | Двухсторонние *UDP/TCP*-потоки | - | - |
| 18 | [25] | Двухсторонние *UDP/TCP*-потоки | Протокол транспортного уровня, порты источника и назначения, среднее число пакетов в потоке, средняя пропускная способность в пакетах/с, байт/с | Категории: *FTP, HTTP, instant messaging*, потоковое видео и *P2P* |
| 19 | [26] | Однонаправленные *TCP/UDP*-потоки | Интервал между временем поступления пакетов, длина пакета, номер порта источника, транспортный протокол, параметр Херста | Категории: голос (*Skype, QQ, WeChat*), видео (*Youtube, Youku, Vimeo, PPStream*), данные (*FTP, Dropbox, Torrent*), интерактивные данные (*LOL, Dota, Http Service*) |
| 20 | [27] | н/д | Минимальное, максимальное, среднее значение, СКО для размеров окон | Приложения: *Minecraft, VLC, Quake 3, VOIP* |

Таблица П.3. Часть 3: Классификация трафика

| № | Статья | Классификация трафика | | |
|---|---|---|---|---|
| | | Методы классификации | Инструменты для интеллектуальной обработки данных | Оценка работоспособности алгоритма |
| 1 | [10] | Методы решающих деревьев, наивный байесовский классификатор и метод опорных векторов (*SVM*) | *Scikit-learn* | Для обучения модели одному классу: 10 первых пакетов, 60 потоков. Классификация по первым 10 пакетам |
| 2 | [11] | *SVM*, метод *K-средних* | - | - |
| 3 | [12] | Нейронные сети *Multilayer Perceptron (MLP), Radial Basis Function (RBF), C4.5*, Наивный Байес, *Belief Network* | *Weka, MATLAB* | 2 секунды потока для классификации, без учета установления сессии |
| 4 | [7] | Наивный Баейс, деревья решений, *LDA*, нейронные сети и *SVM* | Язык программирования *R* | - |
| 5 | [13] | Случайный лес (*Random Forest*) совместно с *DPI* и классификацией по портам | *Scikit-learn* | 20 пакетов для построения модели |
| 6 | [14] | - | - | - |
| 7 | [15] | APPR [16] совместно с информацией, полученной с помощью *DNS*-запросов | *Weka* | 0,1 с для классификации, 10 с для обучения |
| 8 | [16] | *C4.5, PART*, Наивный Байес, *zeroR* и *oneR* | *Weka* | 20 первых пакетов для классификации, не менее 10 потоков и 120 сек для построения модели |
| 9 | [17] | Решающие деревья *J4.8, PART, RIPPER*, Наивный Байес | - | 30 с для классификации |
| 10 | [18] | *SVM* | - | - |
| 11 | [19] | *GMM / EM* | - | - |

| № | Статья | Классификация трафика | | |
|---|---|---|---|---|
| | | Методы классификации | Инструменты для интеллектуальной обработки данных | Оценка работоспособности алгоритма |
| 12 | [6] | Случайный лес *(RF), Stochastic Gradient Boosting (SGB)* и *Extreme Gradient Boosting (XGB)* | Язык программирования *R* | - |
| 13 | [20] | Глубокое обучение и *SVM* | *Weka* и *MATLAB* | - |
| 14 | [21] | Нейронные сети: *AutoEncoder NN (Neural Network), CNN (Convolutional NN)* | *Python*: библиотека *Keras* с *TensorFlow* | - |
| 15 | [22] | *C4.5*, Наивный Байес, Метод ближайших соседей (*k-NN*), *RFB*, сети Байеса | *Weka* | - |
| 16 | [23] | *SVM* и *k-means* кластеризация | *Weka* | - |
| 17 | [24] | глубокое обучение | *Matlab* | 20 секунд потока для обучения |
| 18 | [25] | Нейронные сети с *feedforward, MLP, NARX (Levenberg-Marquardt),* Наивный Байес | - | - |
| 19 | [26] | *DPI* и стеккинг классических методов обучения: *SVM*, Наивный Байес и *kNN* | - | - |
| 20 | [27] | *J.48, SVM+SVO, RF*, Наивный Байес, *Ibk* | *Weka* | 30 пакетов для обучения |

Приложение Б. Результаты классификации при разных способах предварительной обработки данных

Таблица П.4. ***Точность*** *выделения* ***классов*** при классификации методом *Random Forest* для *TCP*-приложений при разных методах предварительной обработки данных

| Методы предобработки данных | *Apple* | *DNS* | *HTTP* | *HTTP_Proxy* | *IMAP* | *IMAPS* | *POPS* | *RTMP* | *SMTP* | *SSH* | *SSL* | *Skype* | *Telnet* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *base data* | 0,727 | 0,993 | 0,824 | 0,942 | 0,854 | 0,720 | 0,832 | 0,728 | 0,911 | 0,966 | 0,700 | 0,740 | 0,925 |
| *Out* | 0,793 | 1,000 | 0,797 | 0,927 | 0,828 | 0,786 | 0,841 | 0,747 | 0,904 | 0,979 | 0,716 | 1,000 | 1,000 |
| *out, 3 STD* | 0,741 | 0,997 | 0,821 | 0,943 | 0,863 | 0,701 | 0,834 | 0,728 | 0,906 | 0,963 | 0,694 | 0,724 | 0,997 |
| *standart* | 0,719 | 0,993 | 0,826 | 0,939 | 0,849 | 0,728 | 0,822 | 0,728 | 0,926 | 0,966 | 0,720 | 0,726 | 0,920 |
| *standart+out* | 0,789 | 1,000 | 0,824 | 0,921 | 0,828 | 0,788 | 0,843 | 0,757 | 0,904 | 0,979 | 0,711 | 1,000 | 1,000 |
| *minmax* | 0,731 | 0,970 | 0,820 | 0,920 | 0,864 | 0,708 | 0,788 | 0,727 | 0,913 | 0,957 | 0,599 | 0,747 | 0,923 |
| *minmax+out* | 0,802 | 1,000 | 0,813 | 0,921 | 0,834 | 0,777 | 0,843 | 0,744 | 0,912 | 0,979 | 0,718 | 1,000 | 1,000 |
| *robust* | 0,729 | 0,993 | 0,827 | 0,942 | 0,858 | 0,720 | 0,841 | 0,725 | 0,911 | 0,969 | 0,695 | 0,731 | 0,928 |
| *robust+out* | 0,793 | 1,000 | 0,798 | 0,927 | 0,828 | 0,786 | 0,841 | 0,747 | 0,904 | 0,979 | 0,719 | 1,000 | 1,000 |
| *power* | 0,731 | 0,993 | 0,823 | 0,945 | 0,843 | 0,720 | 0,822 | 0,734 | 0,924 | 0,963 | 0,685 | 0,739 | 0,925 |
| *power+out* | 0,791 | 1,000 | 0,815 | 0,930 | 0,827 | 0,779 | 0,826 | 0,741 | 0,906 | 0,979 | 0,735 | 0,997 | 1,000 |
| *quantile* | 0,741 | 0,997 | 0,819 | 0,932 | 0,845 | 0,712 | 0,827 | 0,736 | 0,915 | 0,969 | 0,678 | 0,730 | 0,920 |
| *quantile+out* | 0,788 | 1,000 | 0,805 | 0,933 | 0,827 | 0,767 | 0,852 | 0,777 | 0,904 | 0,986 | 0,761 | 0,997 | 1,000 |
| *normalizer* | 0,707 | 0,943 | 0,789 | 0,922 | 0,845 | 0,663 | 0,783 | 0,698 | 0,862 | 0,949 | 0,609 | 0,648 | 0,851 |
| *normalizer+out* | 0,775 | 1,000 | 0,769 | 0,888 | 0,827 | 0,704 | 0,823 | 0,703 | 0,846 | 0,969 | 0,687 | 0,986 | 0,984 |

Таблица П.5. ***Точность*** *выделения* ***классов*** при классификации методом *XGBoost* для *TCP*-приложений при разных методах предварительной обработки данных

| Методы предобработки данных | *Apple* | *DNS* | *HTTP* | *HTTP_Proxy* | *IMAP* | *IMAPS* | *POPS* | *RTMP* | *SMTP* | *SSH* | *SSL* | *Skype* | *Telnet* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *base data* | 0,777 | 0,997 | 0,861 | 0,930 | 0,854 | 0,744 | 0,794 | 0,759 | 0,930 | 0,966 | 0,679 | 0,757 | 0,934 |
| *Out* | 0,825 | 1,000 | 0,861 | 0,931 | 0,866 | 0,830 | 0,835 | 0,806 | 0,927 | 0,966 | 0,779 | 1,000 | 1,000 |
| *out, 3 STD* | 0,762 | 0,997 | 0,877 | 0,934 | 0,895 | 0,764 | 0,804 | 0,767 | 0,945 | 0,967 | 0,691 | 0,753 | 0,993 |
| *standart* | 0,779 | 0,997 | 0,840 | 0,926 | 0,849 | 0,739 | 0,783 | 0,770 | 0,927 | 0,969 | 0,665 | 0,770 | 0,940 |
| *standart+out* | 0,823 | 1,000 | 0,856 | 0,913 | 0,862 | 0,827 | 0,845 | 0,836 | 0,921 | 0,973 | 0,766 | 1,000 | 1,000 |
| *minmax* | 0,774 | 0,997 | 0,859 | 0,927 | 0,848 | 0,753 | 0,801 | 0,762 | 0,939 | 0,969 | 0,647 | 0,750 | 0,937 |
| *minmax+out* | 0,815 | 1,000 | 0,866 | 0,935 | 0,877 | 0,823 | 0,827 | 0,823 | 0,924 | 0,979 | 0,767 | 1,000 | 1,000 |
| *robust* | 0,789 | 0,997 | 0,833 | 0,926 | 0,854 | 0,734 | 0,792 | 0,755 | 0,935 | 0,966 | 0,649 | 0,769 | 0,934 |
| *robust+out* | 0,823 | 1,000 | 0,849 | 0,918 | 0,868 | 0,828 | 0,941 | 0,822 | 0,921 | 0,963 | 0,813 | 1,000 | 1,000 |
| *power* | 0,769 | 0,993 | 0,835 | 0,920 | 0,853 | 0,736 | 0,806 | 0,756 | 0,921 | 0,969 | 0,681 | 0,747 | 0,937 |
| *power+out* | 0,813 | 1,000 | 0,881 | 0,941 | 0,877 | 0,828 | 0,834 | 0,812 | 0,924 | 0,973 | 0,778 | 1,000 | 1,000 |
| *quantile* | 0,778 | 0,993 | 0,852 | 0,917 | 0,860 | 0,764 | 0,803 | 0,782 | 0,929 | 0,969 | 0,690 | 0,768 | 0,937 |
| *quantile+out* | 0,813 | 1,000 | 0,875 | 0,922 | 0,880 | 0,854 | 0,829 | 0,830 | 0,932 | 0,973 | 0,802 | 1,000 | 1,000 |
| *normalizer* | 0,723 | 0,974 | 0,777 | 0,937 | 0,839 | 0,650 | 0,762 | 0,732 | 0,884 | 0,957 | 0,569 | 0,691 | 0,880 |
| *normalizer+out* | 0,772 | 0,993 | 0,797 | 0,896 | 0,858 | 0,744 | 0,795 | 0,755 | 0,898 | 0,973 | 0,685 | 0,983 | 0,997 |

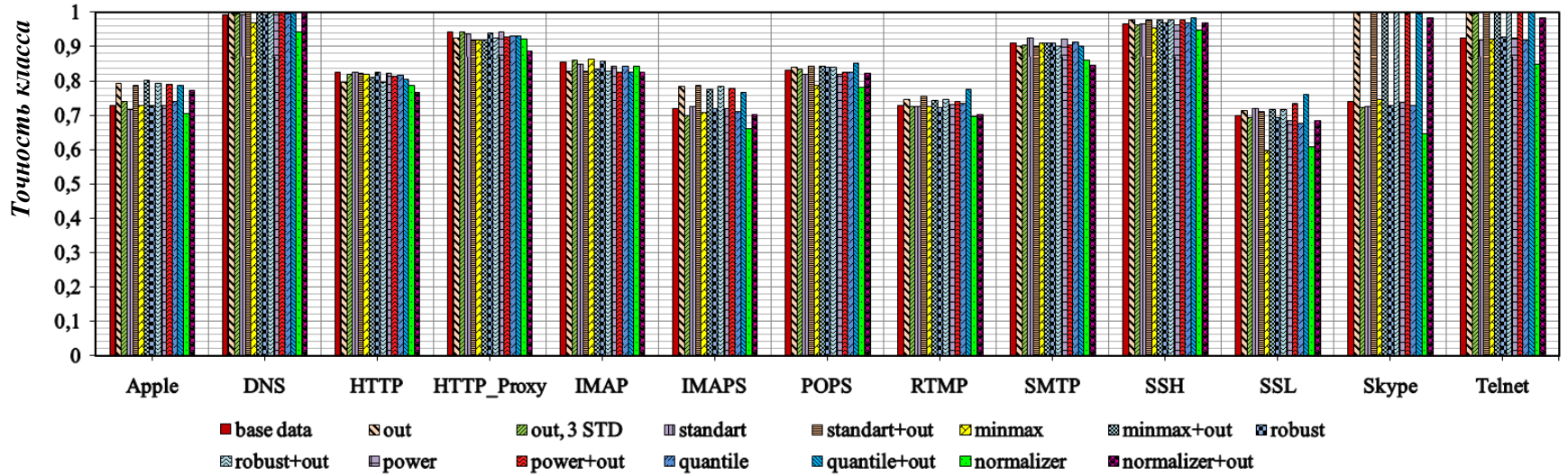



Рисунок П.1 - ***Точность*** классификации методом *Random Forest* и разных методах предварительной обработки

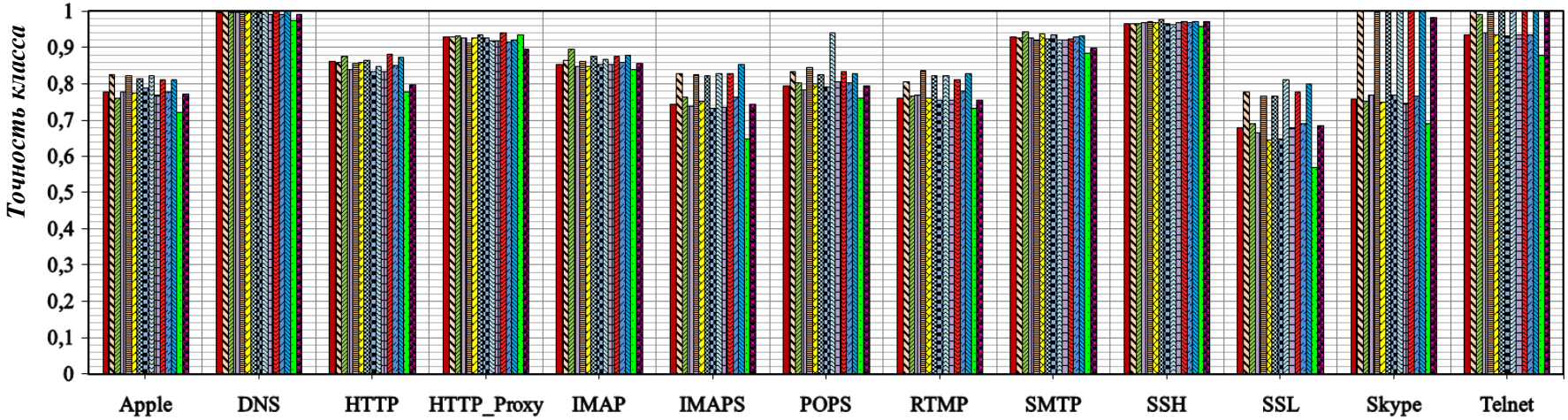



Рисунок П.2 - ***Точность*** классификации методом *XGBoost* и разных методах предварительной обработки

Таблица П.6. ***Полнота*** классификации методом *Random Forest* для *TCP*-приложений при разных методах предварительной обработки данных

| Методы предобработки данных | *Apple* | *DNS* | *HTTP* | *HTTP_Proxy* | *IMAP* | *IMAPS* | *POPS* | *RTMP* | *SMTP* | *SSH* | *SSL* | *Skype* | *Telnet* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *base data* | 0,887 | 1,000 | 0,750 | 0,923 | 0,900 | 0,813 | 0,710 | 0,787 | 0,960 | 0,947 | 0,490 | 0,713 | 0,990 |
| *Out* | 0,973 | 1,000 | 0,720 | 0,930 | 0,880 | 0,797 | 0,703 | 0,837 | 0,970 | 0,530 | 0,563 | 1,000 | 1,000 |
| *out, 3 STD* | 0,927 | 1,000 | 0,717 | 0,943 | 0,907 | 0,827 | 0,703 | 0,783 | 0,967 | 0,957 | 0,483 | 0,700 | 0,997 |
| *standart* | 0,897 | 1,000 | 0,743 | 0,923 | 0,900 | 0,803 | 0,707 | 0,783 | 0,967 | 0,947 | 0,497 | 0,707 | 0,990 |
| *standart+out* | 0,973 | 1,000 | 0,720 | 0,933 | 0,883 | 0,803 | 0,700 | 0,850 | 0,970 | 0,950 | 0,567 | 1,000 | 1,000 |
| *minmax* | 0,880 | 1,000 | 0,743 | 0,923 | 0,887 | 0,783 | 0,712 | 0,763 | 0,943 | 0,957 | 0,503 | 0,717 | 0,993 |
| *minmax+out* | 0,973 | 1,000 | 0,723 | 0,933 | 0,887 | 0,790 | 0,717 | 0,843 | 0,963 | 0,950 | 0,570 | 1,000 | 1,000 |
| *robust* | 0,887 | 1,000 | 0,750 | 0,927 | 0,903 | 0,813 | 0,703 | 0,790 | 0,960 | 0,950 | 0,493 | 0,707 | 0,990 |
| *robust+out* | 0,973 | 1,000 | 0,723 | 0,930 | 0,550 | 0,797 | 0,703 | 0,837 | 0,970 | 0,953 | 0,563 | 1,000 | 1,000 |
| *power* | 0,887 | 1,000 | 0,743 | 0,917 | 0,910 | 0,797 | 0,710 | 0,780 | 0,967 | 0,950 | 0,493 | 0,717 | 0,990 |
| *power+out* | 0,970 | 1,000 | 0,720 | 0,923 | 0,873 | 0,797 | 0,713 | 0,840 | 0,967 | 0,950 | 0,573 | 1,000 | 1,000 |
| *quantile* | 0,887 | 1,000 | 0,740 | 0,913 | 0,890 | 0,817 | 0,703 | 0,790 | 0,967 | 0,950 | 0,483 | 0,703 | 0,993 |
| *quantile+out* | 0,967 | 1,000 | 0,730 | 0,927 | 0,860 | 0,780 | 0,730 | 0,860 | 0,970 | 0,940 | 0,627 | 1,000 | 1,000 |
| *normalizer* | 0,813 | 0,987 | 0,733 | 0,907 | 0,986 | 0,760 | 0,697 | 0,687 | 0,957 | 0,937 | 0,390 | 0,663 | 0,933 |
| *normalizer+out* | 0,907 | 0,983 | 0,710 | 0,923 | 0,860 | 0,760 | 0,697 | 0,773 | 0,937 | 0,923 | 0,527 | 0,963 | 0,997 |

Таблица П.7. ***Полнота*** классификации методом *XGBoost* для *TCP*-приложений при разных методах предварительной обработки данных

| Методы предобработки данных | *Apple* | *DNS* | *HTTP* | *HTTP_Proxy* | *IMAP* | *IMAPS* | *POPS* | *RTMP* | *SMTP* | *SSH* | *SSL* | *Skype* | *Telnet* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *base data* | 0,897 | 1,000 | 0,767 | 0,930 | 0,893 | 0,767 | 0,760 | 0,797 | 0,970 | 0,943 | 0,537 | 0,757 | 0,990 |
| *Out* | 0,960 | 1,000 | 0,787 | 0,950 | 0,903 | 0,813 | 0,777 | 0,873 | 0,970 | 0,957 | 0,647 | 1,000 | 1,000 |
| *out, 3 STD* | 0,907 | 1,000 | 0,783 | 0,943 | 0,913 | 0,830 | 0,753 | 0,780 | 0,977 | 0,970 | 0,567 | 0,740 | 0,997 |
| *standart* | 0,883 | 1,000 | 0,750 | 0,920 | 0,900 | 0,753 | 0,757 | 0,803 | 0,970 | 0,950 | 0,534 | 0,760 | 0,993 |
| *standart+out* | 0,963 | 1,000 | 0,790 | 0,947 | 0,893 | 0,813 | 0,780 | 0,847 | 0,970 | 0,963 | 0,667 | 1,000 | 1,000 |
| *minmax* | 0,877 | 1,000 | 0,750 | 0,930 | 0,893 | 0,773 | 0,763 | 0,800 | 0,970 | 0,953 | 0,537 | 0,750 | 0,990 |
| *minmax+out* | 0,957 | 1,000 | 0,800 | 0,953 | 0,903 | 0,823 | 0,767 | 0,870 | 0,973 | 0,953 | 0,647 | 1,000 | 1,000 |
| *robust* | 0,887 | 1,000 | 0,750 | 0,920 | 0,900 | 0,780 | 0,747 | 0,760 | 0,967 | 0,947 | 0,587 | 0,720 | 0,990 |
| *robust+out* | 0,963 | 1,000 | 0,787 | 0,937 | 0,897 | 0,820 | 0,777 | 0,877 | 0,973 | 0,957 | 0,667 | 1,000 | 1,000 |
| *power* | 0,890 | 1,000 | 0,743 | 0,917 | 0,887 | 0,770 | 0,750 | 0,787 | 0,977 | 0,950 | 0,540 | 0,747 | 0,937 |
| *power+out* | 0,957 | 1,000 | 0,790 | 0,953 | 0,900 | 0,833 | 0,770 | 0,877 | 0,967 | 0,963 | 0,653 | 1,000 | 1,000 |
| *quantile* | 0,887 | 1,000 | 0,770 | 0,920 | 0,900 | 0,777 | 0,787 | 0,800 | 0,967 | 0,950 | 0,557 | 0,763 | 0,993 |
| *quantile+out* | 0,957 | 1,000 | 0,793 | 0,943 | 0,903 | 0,826 | 0,773 | 0,897 | 0,967 | 0,960 | 0,700 | 1,000 | 1,000 |
| *normalizer* | 0,793 | 0,993 | 0,733 | 0,890 | 0,850 | 0,743 | 0,703 | 0,700 | 0,937 | 0,953 | 0,480 | 0,670 | 0,953 |
| *normalizer+out* | 0,893 | 0,990 | 0,733 | 0,923 | 0,863 | 0,783 | 0,683 | 0,810 | 0,967 | 0,950 | 0,587 | 0,973 | 1,000 |

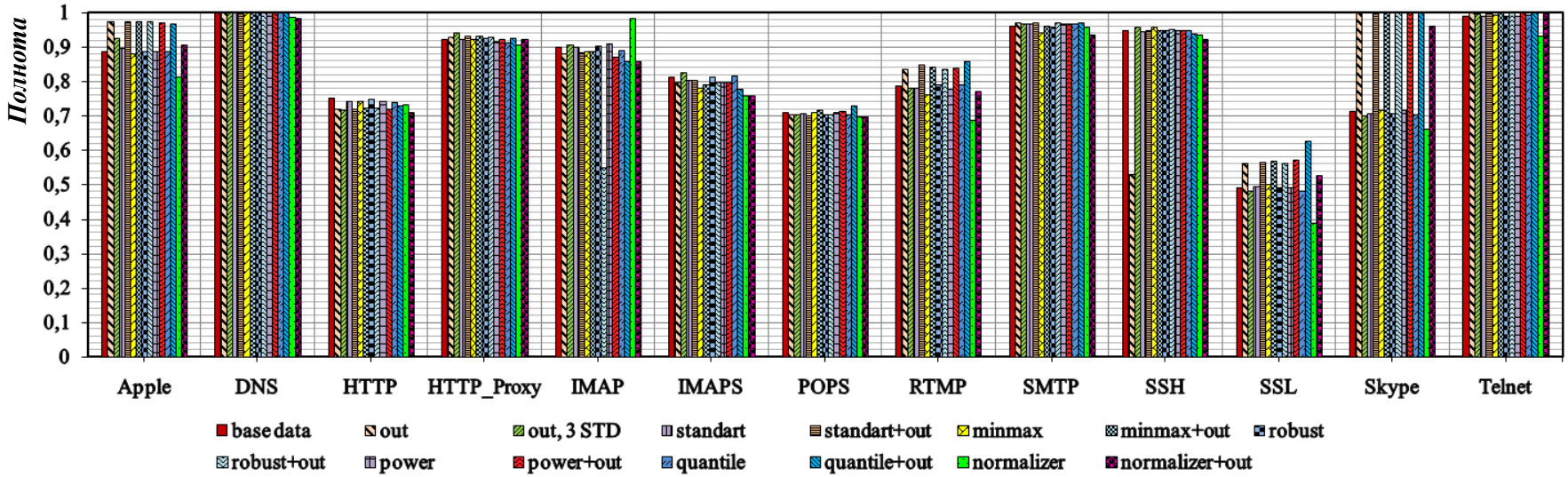



Рисунок П.3 - ***Полнота*** классификации методом *Random Forest* и разных методах предварительной обработки

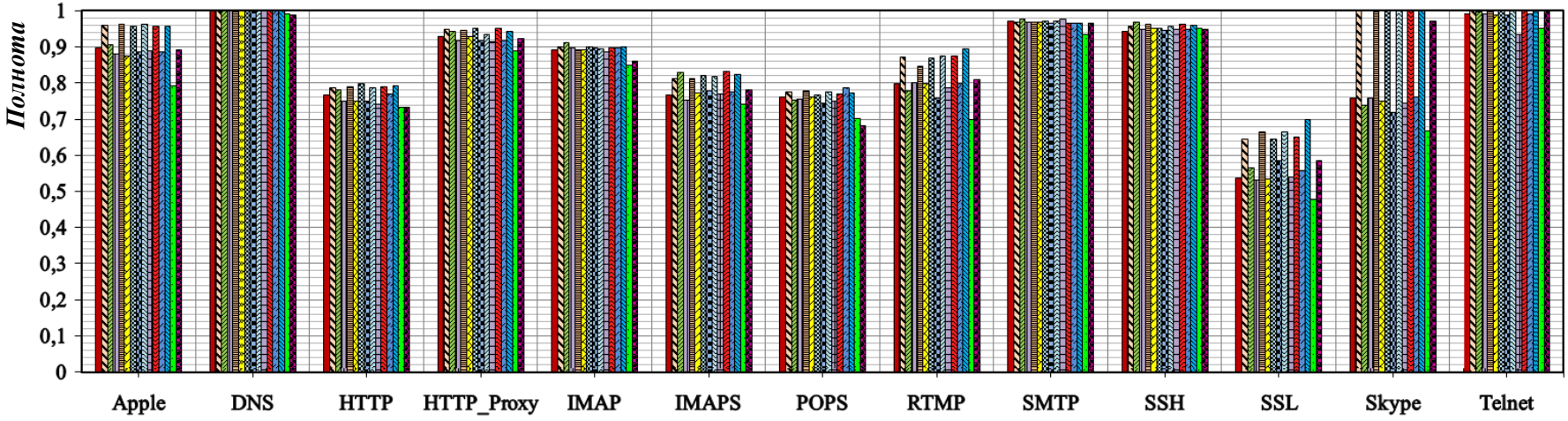



Рисунок П.4 - ***Полнота*** классификации методом *XGBoost* и разных методах предварительной обработки

Таблица П.8. ***F1-мера*** классификации методом *Random Forest* для *TCP* - приложений при разных методах предварительной обработки данных

| Методы предобработки данных | *Apple* | *DNS* | *HTTP* | *HTTP_Proxy* | *IMAP* | *IMAPS* | *POPS* | *RTMP* | *SMTP* | *SSH* | *SSL* | *Skype* | *Telnet* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *base data* | 0,799 | 0,997 | 0,785 | 0,933 | 0,877 | 0,764 | 0,766 | 0,756 | 0,935 | 0,956 | 0,576 | 0,727 | 0,957 |
| *Out* | 0,874 | 1,000 | 0,757 | 0,928 | 0,853 | 0,791 | 0,766 | 0,789 | 0,936 | 0,966 | 0,631 | 1,000 | 1,000 |
| *out, 3 STD* | 0,824 | 0,998 | 0,765 | 0,943 | 0,885 | 0,768 | 0,763 | 0,754 | 0,935 | 0,960 | 0,570 | 0,712 | 0,997 |
| *standart* | 0,798 | 0,997 | 0,782 | 0,931 | 0,873 | 0,764 | 0,760 | 0,754 | 0,947 | 0,956 | 0,588 | 0,716 | 0,953 |
| *standart+out* | 0,872 | 1,000 | 0,769 | 0,927 | 0,855 | 0,795 | 0,765 | 0,800 | 0,936 | 0,964 | 0,631 | 1,000 | 1,000 |
| *minmax* | 0,799 | 0,998 | 0,780 | 0,922 | 0,875 | 0,744 | 0,750 | 0,745 | 0,928 | 0,957 | 0,585 | 0,721 | 0,957 |
| *minmax+out* | 0,880 | 1,000 | 0,765 | 0,927 | 0,859 | 0,783 | 0,774 | 0,791 | 0,937 | 0,964 | 0,636 | 1,000 | 1,000 |
| *robust* | 0,800 | 0,997 | 0,787 | 0,934 | 0,880 | 0,764 | 0,766 | 0,756 | 0,935 | 0,960 | 0,577 | 0,719 | 0,958 |
| *robust+out* | 0,874 | 1,000 | 0,759 | 0,928 | 0,853 | 0,791 | 0,766 | 0,789 | 0,936 | 0,966 | 0,632 | 1,000 | 1,000 |
| *power* | 0,801 | 0,997 | 0,781 | 0,931 | 0,875 | 0,756 | 0,762 | 0,756 | 0,945 | 0,956 | 0,574 | 0,728 | 0,957 |
| *power+out* | 0,871 | 1,000 | 0,765 | 0,926 | 0,849 | 0,787 | 0,766 | 0,788 | 0,935 | 0,964 | 0,644 | 0,998 | 1,000 |
| *quantile* | 0,807 | 0,998 | 0,778 | 0,923 | 0,867 | 0,761 | 0,760 | 0,762 | 0,940 | 0,960 | 0,564 | 0,716 | 0,955 |
| *quantile+out* | 0,868 | 1,000 | 0,766 | 0,930 | 0,843 | 0,774 | 0,786 | 0,816 | 0,936 | 0,962 | 0,687 | 0,998 | 1,000 |
| *normalizer* | 0,757 | 0,964 | 0,760 | 0,914 | 0,850 | 0,708 | 0,737 | 0,692 | 0,907 | 0,943 | 0,476 | 0,656 | 0,890 |
| *normalizer+out* | 0,836 | 0,992 | 0,738 | 0,905 | 0,843 | 0,731 | 0,755 | 0,737 | 0,889 | 0,945 | 0,596 | 0,975 | 0,990 |

Таблица П.9. ***F1-мера*** классификации методом *XGBoost* для *TCP*-приложений при разных методах предварительной обработки данных

| Методы предобработки данных | *Apple* | *DNS* | *HTTP* | *HTTP_Proxy* | *IMAP* | *IMAPS* | *POPS* | *RTMP* | *SMTP* | *SSH* | *SSL* | *Skype* | *Telnet* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *base data* | 0,833 | 0,998 | 0,811 | 0,930 | 0,873 | 0,755 | 0,777 | 0,777 | 0,949 | 0,954 | 0,600 | 0,757 | 0,961 |
| *Out* | 0,888 | 1,000 | 0,822 | 0,941 | 0,884 | 0,822 | 0,805 | 0,838 | 0,948 | 0,961 | 0,707 | 1,000 | 1,000 |
| *out, 3 STD* | 0,828 | 0,998 | 0,827 | 0,934 | 0,904 | 0,796 | 0,778 | 0,774 | 0,961 | 0,968 | 0,623 | 0,746 | 0,995 |
| *standart* | 0,828 | 0,998 | 0,792 | 0,923 | 0,874 | 0,746 | 0,769 | 0,786 | 0,948 | 0,960 | 0,598 | 0,765 | 0,966 |
| *standart+out* | 0,888 | 1,000 | 0,821 | 0,930 | 0,877 | 0,820 | 0,811 | 0,841 | 0,945 | 0,968 | 0,713 | 1,000 | 1,000 |
| *minmax* | 0,822 | 0,998 | 0,801 | 0,928 | 0,870 | 0,763 | 0,782 | 0,780 | 0,954 | 0,961 | 0,587 | 0,750 | 0,963 |
| *minmax+out* | 0,880 | 1,000 | 0,832 | 0,944 | 0,890 | 0,823 | 0,796 | 0,846 | 0,948 | 0,966 | 0,702 | 1,000 | 1,000 |
| *robust* | 0,835 | 0,998 | 0,789 | 0,923 | 0,877 | 0,756 | 0,768 | 0,757 | 0,951 | 0,956 | 0,616 | 0,743 | 0,961 |
| *robust+out* | 0,888 | 1,000 | 0,817 | 0,927 | 0,882 | 0,824 | 0,808 | 0,848 | 0,947 | 0,960 | 0,733 | 1,000 | 1,000 |
| *power* | 0,825 | 0,997 | 0,787 | 0,918 | 0,869 | 0,752 | 0,777 | 0,771 | 0,948 | 0,960 | 0,602 | 0,747 | 0,964 |
| *power+out* | 0,879 | 1,000 | 0,833 | 0,947 | 0,888 | 0,831 | 0,801 | 0,843 | 0,945 | 0,968 | 0,710 | 1,000 | 1,000 |
| *quantile* | 0,829 | 0,997 | 0,809 | 0,918 | 0,879 | 0,770 | 0,795 | 0,791 | 0,948 | 0,960 | 0,616 | 0,766 | 0,964 |
| *quantile+out* | 0,879 | 1,000 | 0,832 | 0,932 | 0,891 | 0,835 | 0,800 | 0,862 | 0,949 | 0,966 | 0,747 | 1,000 | 1,000 |
| *normalizer* | 0,757 | 0,984 | 0,755 | 0,913 | 0,844 | 0,694 | 0,731 | 0,716 | 0,909 | 0,955 | 0,521 | 0,680 | 0,915 |
| *normalizer+out* | 0,828 | 0,992 | 0,764 | 0,910 | 0,860 | 0,763 | 0,735 | 0,781 | 0,931 | 0,961 | 0,632 | 0,978 | 0,998 |

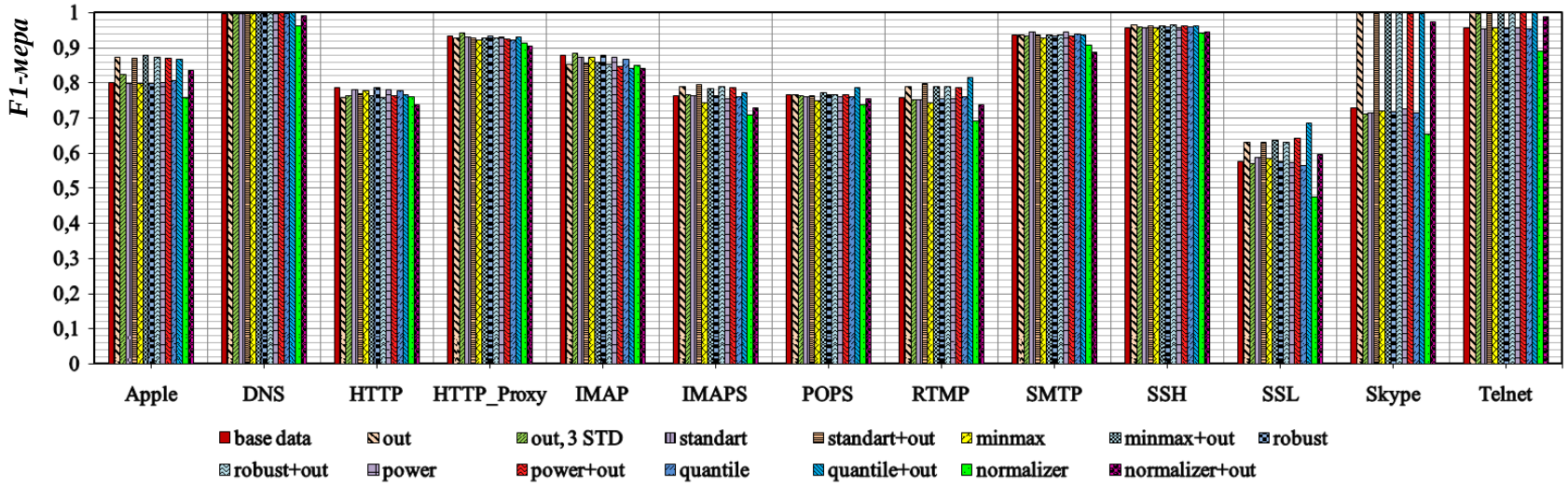



Рисунок П.5 - ***F1-мера*** классификации методом *Random Forest* и разных методах предварительной обработки

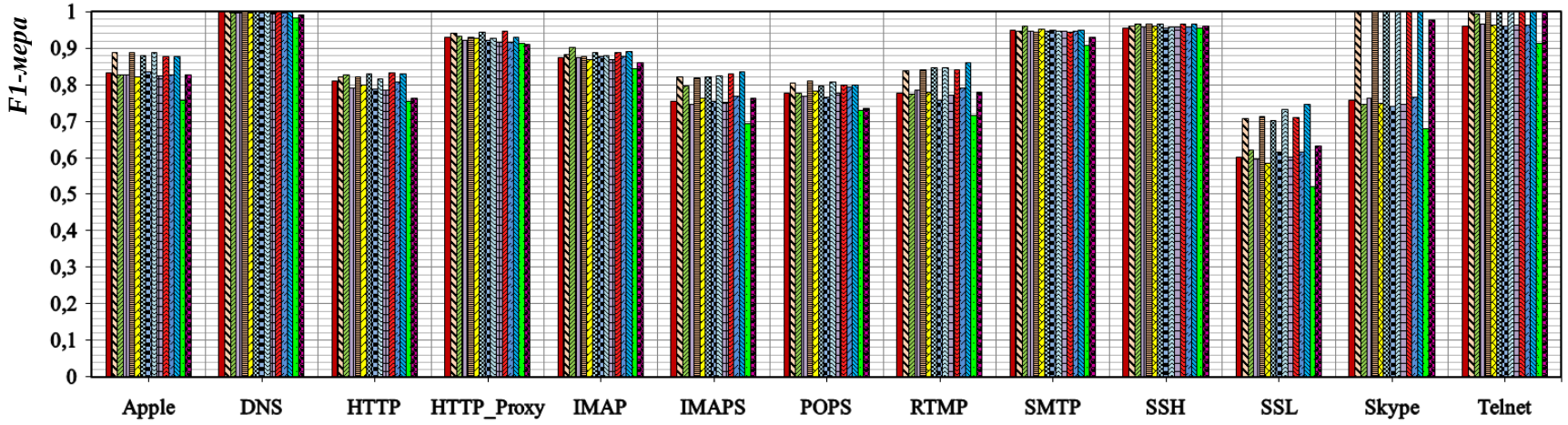



Рисунок П.6 - ***F1-мера*** классификации методом *XGBoost* и разных методах предварительной обработки

# Приложение В. Листинг программы разработанного P4-коммутатора

Описание вспомогательных переменных:

```
typedef bit<9> egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
```

Описание заголовков Ethernet *II*, *IPv4*, *TCP* и *UDP*:

```
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;}
header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;}
header tcp_t {
    bit<16> srcPort;
    bit<16> dstPort;
    bit<32> seqNo;
    bit<32> ackNo;
    bit<4>  dataOffset;
    bit<4>  res;
    bit<8>  flags;
    bit<16> window;
    bit<16> checksum;
    bit<16> urgentPtr;}
header udp_t {
    bit<16> srcPort;
    bit<16> dstPort;
    bit<16> length_;
    bit<16> checksum;}
struct metadata {bit<32> packetID_meta;}
struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
    tcp_t        tcp;
    udp_t        udp;}
}
struct digest_new_flow {
    ip4Addr_t   srcAddr;
    ip4Addr_t   dstAddr;
    bit<8>      protocol;
    bit<16>     srcPort_new_flow;
    bit<16>     dstPort_new_flow;
    bit<32>     order_new_flow;
    bit<48>     time [0-14];
    bit<32>     length [0-14];}
```

Задание постоянных значений:

```
const bit<16> TYPE_IPV4 = 0x800;
const bit<8> TYPE_UDP = 0x11;
const bit<8> TYPE_TCP = 0x6;
const bit<32> NUMBER_OF_FLOW = 0x3e8;
const bit<32> SIZE_OF_FLOWS = 0x00f;
const bit<32> SIZE_OF_FLOW = NUMBER_OF_FLOW * SIZE_OF_FLOWS;
```

Парсер пакетов:

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata)
state start { packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
           0x0800: parse_ipv4;
        default: accept; }}
//распарсивание IPv4 - заголовка
state parse_ipv4 {packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.protocol) {
        TYPE_UDP: parse_udp;
        TYPE_TCP: parse_tcp;
        default: accept; }}
//распарсивание UDP - заголовка
state parse_udp {packet.extract(hdr.udp);
        transition accept; }
//распарсивание TCP - заголовка
state parse_tcp {packet.extract(hdr.tcp);
        transition accept; }
```

Проверка контрольной суммы:

```
control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }}
```

Блок обработки пакета:

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
```

Описание регистров:

```
    register<bit<32>>(1) flowID_digest;
    register<bit<32>>(NUMBER_OF_FLOW) srcAddr_digest;
    register<bit<32>>(NUMBER_OF_FLOW) dstAddr_digest;
    register<bit<8>>(NUMBER_OF_FLOW) protocol_digest;
    register<bit<16>>(NUMBER_OF_FLOW) srcPort_digest;
    register<bit<16>>(NUMBER_OF_FLOW) dstPort_digest;
    register<bit<32>>(NUMBER_OF_FLOW) packet_counter_digest;
    register<bit<48>>(SIZE_OF_FLOW) ingress_global_timestamp_digest;
    register<bit<32>>(SIZE_OF_FLOW) packet_length_digest;
    register<bit<32>>(NUMBER_OF_FLOW) packetID_digest;
```

Описание вспомогательных переменных:

```
bit<32> tmp_flowID_digest;
    bit<32> tmp_srcAddr_digest;
    bit<32> tmp_dstAddr_digest;
    bit<8> tmp_protocol_digest;
    bit<16> tmp_srcPort_digest;
    bit<16> tmp_dstPort_digest;
    bit<32> tmp_packet_counter_digest;
    bit<32> tmp_packetID_digest;
    bit<16> tmp_srcPort_in_packet_digest;
    bit<16> tmp_dstPort_in_packet_digest;
```

```
    bit<2> marker; bit<32> n; bit<32> n1; bit<2> q;
```

Маршрутизация трафика:

```
    action drop() {mark_to_drop();}
    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl=hdr.ipv4.ttl-1;
    standard_metadata.egress_spec = port; }
```

Регистрация пакетов из новых потоков:

```
action new_flow_action ()
        {   flowID_digest.read(tmp_flowID_digest, 0);
            packet_counter_digest.read(tmp_packet_counter_digest,
            tmp_flowID_digest);
            tmp_packet_counter_digest=1;
            packet_counter_digest.write(tmp_flowID_digest, (bit<32>)
            tmp_packet_counter_digest);
            srcAddr_digest.write(tmp_flowID_digest, (bit<32>)hdr.ipv4.srcAddr);
            dstAddr_digest.write(tmp_flowID_digest, (bit<32>)hdr.ipv4.dstAddr);
            protocol_digest.write(tmp_flowID_digest, (bit<8>)hdr.ipv4.protocol);
            srcPort_digest.write(tmp_flowID_digest, (bit<16>)
            tmp_srcPort_in_packet_digest);
            dstPort_digest.write(tmp_flowID_digest, (bit<16>)
            tmp_dstPort_in_packet_digest);
            bit<32> tmp_flowID_for_packet_digest;
            tmp_flowID_for_packet_digest=tmp_flowID_digest*SIZE_OF_FLOWS;
ingress_global_timestamp_digest.write(tmp_flowID_for_packet_digest,(bit<48>)standard_metad
ata.ingress_global_timestamp);
            meta.packetID_meta= tmp_flowID_for_packet_digest;
            packet_length_digest.write(tmp_flowID_for_packet_digest,
(bit<32>)standard_metadata.packet_length);
            tmp_flowID_digest=tmp_flowID_digest+32w1;
            tmp_flowID_digest = (tmp_flowID_digest == NUMBER_OF_FLOW) ? 0 :
            tmp_flowID_digest;
            flowID_digest.write(0, tmp_flowID_digest);}
```

Регистрация пакетов из старых потоков:

```
action old_flow_action (bit<32> order_flow)
        {marker=1;
        packetID_digest.read(tmp_packetID_digest, order_flow);
        tmp_packetID_digest=tmp_packetID_digest+32w1;
        tmp_packetID_digest = (tmp_packetID_digest == SIZE_OF_FLOWS) ? 0
:tmp_packetID_digest;
        packetID_digest.write(order_flow, tmp_packetID_digest);
        tmp_packetID_digest=order_flow*SIZE_OF_FLOWS+tmp_packetID_digest;
        ingress_global_timestamp_digest.write(tmp_packetID_digest,
(bit<48>)standard_metadata.ingress_global_timestamp);
    meta.packetID_meta= tmp_packetID_digest;
packet_length_digest.write(tmp_packetID_digest,(bit<32>)standard_metadata.packet_length);
        packet_counter_digest.read(tmp_packet_counter_digest, order_flow);
        if (tmp_packet_counter_digest=15)
        {if (tmp_protocol_digest)== TYPE_TCP { digest<digest_new_flow>
(1,{hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.ipv4.protocol, tmp_srcPort_in_packet_digest,
tmp_dstPort_in_packet_digest, tmp_flowID_digest,
standard_metadata.ingress_global_timestamp_digest [0-14], packet_length_digest [0-14]});
if (tmp_protocol_digest)== TYPE_UDP { digest<digest_new_flow>
(1,{hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.ipv4.protocol, tmp_srcPort_in_packet_digest,
tmp_dstPort_in_packet_digest, tmp_flowID_digest,
standard_metadata.ingress_global_timestamp_digest [0-9], packet_length_digest [0-9]});}}
        tmp_packet_counter_digest= tmp_packet_counter_digest+1;
        packet_counter_digest.write(order_flow, (bit<32>)tmp_packet_counter_digest); }
```

Идентификация потока

```
action reserve_action ()
    {
        srcAddr_digest.read(tmp_srcAddr_digest, n);
        dstAddr_digest.read(tmp_dstAddr_digest, n);
        protocol_digest.read(tmp_protocol_digest, n);
        srcPort_digest.read(tmp_srcPort_digest, n);
        dstPort_digest.read(tmp_dstPort_digest, n);
if
((hdr.ipv4.srcAddr==tmp_srcAddr_digest)&&(hdr.ipv4.dstAddr==tmp_dstAddr_digest)&&(hdr.ipv4
.protocol==tmp_protocol_digest)&&(tmp_srcPort_in_packet_digest==tmp_srcPort_digest)&&(tmp_
dstPort_in_packet_digest==tmp_dstPort_digest))
            {q=1;n1=n;}
if (n==0)
               {n=NUMBER_OF_FLOW-1;}
              else {n=n-1;}}
```

Таблица маршрутизации

```
table ipv4_lpm {
        key = {hdr.ipv4.dstAddr: lpm;}
        actions = {
            ipv4_forward;
            drop;
            NoAction;   }
        size = 1024;
        default_action = NoAction();}
```

Алгоритм обработки пакета на входе:

```
apply {
     if (hdr.ipv4.isValid()) {
         if ((hdr.ipv4.protocol==TYPE_TCP)||(hdr.ipv4.protocol==TYPE_UDP))
         { marker=0;
             if (hdr.ipv4.protocol==TYPE_UDP)
                {tmp_srcPort_in_packet_digest=hdr.udp.srcPort;
                tmp_dstPort_in_packet_digest=hdr.udp.dstPort;}
            else
             {if (hdr.ipv4.protocol==TYPE_TCP)
                {tmp_srcPort_in_packet_digest=hdr.tcp.srcPort;
                tmp_dstPort_in_packet_digest=hdr.tcp.dstPort;}}
         if (marker==0)
            { n1=0; q=0;
                flowID_digest.read(tmp_flowID_digest, 0);
                 if (tmp_flowID_digest==0)
                 {n=NUMBER_OF_FLOW-1; }
              else
                 {n=tmp_flowID_digest-1;}

                 reserve_action ();
                 reserve_action ();
                 reserve_action ();
                 reserve_action ();
                 reserve_action ();
                 reserve_action ();
                 reserve_action ();
                 reserve_action ();
                 reserve_action ();
                 reserve_action ();
                 reserve_action ();
                 if (q==1)
                 {old_flow_action (n1);
                 }
                else
```

```
    {new_flow_action ();}}}
        ipv4_lpm.apply();}}}
```

Обработка пакета на выходе коммутатора:

```
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {apply{}}
```

Пересчет контрольной суммы:

```
control MyComputeChecksum(inout headers hdr, inout metadata meta) {
     apply {
 update_checksum(
     hdr.ipv4.isValid(),
            { hdr.ipv4.version,
              hdr.ipv4.ihl,
              hdr.ipv4.diffserv,
              hdr.ipv4.totalLen,
              hdr.ipv4.identification,
              hdr.ipv4.flags,
              hdr.ipv4.fragOffset,
              hdr.ipv4.ttl,
              hdr.ipv4.protocol,
              hdr.ipv4.srcAddr,
              hdr.ipv4.dstAddr },
              hdr.ipv4.hdrChecksum,
              HashAlgorithm.csum16);  }}
```

Блок депарсера:

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
    packet.emit(hdr.ethernet);
    packet.emit(hdr.ipv4);
    packet.emit(hdr.tcp);
    packet.emit(hdr.udp); }}
```

Определение структуры коммутатора:

```
V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

# Приложение Г. Акты о внедрении результатов диссертационной работы

УТВЕРЖДАЮ
Проректор по учебной работе ордена Трудового Красного Знамени федерального государственного бюджетного образовательного учреждения высшего образования «Московский технический университет связи и информатики»
к.т.н., доцент _____ Титов Е.В.
«26» мая 2021 г.

**АКТ**

об использовании результатов диссертационной работы Красновой Ирины Артуровны на тему «Динамическая классификация потоков трафика на основе машинного обучения для обеспечения качества обслуживания в мультисервисной программно-конфигурируемой сети» в учебном процессе кафедры С и СФС МТУСИ

Комиссия в составе заведующей Центром планирования и сопровождения учебного процесса Патенченковой Е.К., декан факультета Сети и системы связи (С и СС), к.т.н. Миронов Ю.Б. и доцента кафедры Сети связи и системы коммутации (СС и СК), к.т.н. Маликовой Е.Е. составила настоящий акт о том, что результаты диссертационной работы Красновой И.А., а именно:

— статическая модель классификации трафика методами машинного обучения для поддержания QoS, работающая на основе матрицы признаков, в которой признаками являются индивидуальные статистические параметры первых 10-15 пакетов;

— алгоритм гибкого сбора статистической информации о пакетах в SDN-сетях

используются в учебном процессе кафедры Си СФС. Форма внедрения: разработка рабочей программы, создание лабораторного стенда кафедры С и СФС, проведение лабораторных занятий дисциплины под названием « Виртуализация сетевых функций и программно-конфигурируемых сетей», направление подготовки: «11.04.02 - Инфокоммуникационные технологии и системы связи», профиль подготовки: «Мультисервисные инфокоммуникационные технологии».

Издано учебное пособие с описанием лабораторных работ, использующихся в учебном процессе кафедры С и СФС:

Краснова. И.А. Виртуализация сетевых функций и программно-конфигурируемые сети: учебное пособие / И.А. Краснова, В.А. Маньков, А.Е. Панов; МТУСИ – Москва, 2020.– 126 с.

**Заведующая Центром планирования и сопровождения учебного процесса** **Патенченкова Е.К.**

**Декан факультета С и СС** **Миронов Ю.Б.**

**Доцент кафедры СС и СК** **Маликова Е.Е.**

**Общество с ограниченной ответственностью**
**научно-производственная фирма «Гранч»**
**(ООО НПФ «Гранч»)**
**Адрес: 630015, г. Новосибирск,**
**ул. Королева 40, корп.1, офис 304**
**Тел./факс: +7 (383) 2-120-316**
**Почта: info@granch.ru**

**УТВЕРЖДАЮ**
**Генеральный директор**
**ООО НПФ «Гранч»**
**_____ О.В. Бочарников**
**«26» мая 2021 г.**

# АКТ

О внедрении результатов диссертационной работы Красновой Ирины Артуровны на тему «Динамическая классификация потоков трафика на основе машинного обучения для обеспечения качества обслуживания в мультисервисной программно-конфигурируемой сети»

Комиссия в составе Бочарникова О.В., Галенчиковой О.А. и Костенко М.В. составила настоящий акт о том, что результаты диссертационной работы Красновой Ирины Артуровны, а именно, «Метод динамической классификации трафика на основе машинного обучения для обеспечения качества обслуживания», используется в проектах сетей связи, разрабатываемых ООО НПФ «Гранч».

Генеральный директор _____ Бочарников О.В.

Зам. начальника НТО _____ Галенчикова О.А.

Ведущий инженер-электроник _____ Костенко М.В.

ИнформИнвестГрупп
InformInvestGroup

**УТВЕРЖДАЮ**
**Генеральный директор**
**ЗАО «ИнформИнвестГрупп»**
**Темников Андрей Викторович**
**«31» мая 2021 г.**

**АКТ**

об использовании результатов диссертационной работы Красновой Ирины Артуровны на тему *«Динамическая классификация потоков трафика на основе машинного обучения для обеспечения качества обслуживания в мультисервисной программно-конфигурируемой сети»*
в ЗАО «ИнформИнвестГрупп»

Комиссия в составе Вексельман М.И, Зайцев Н.Л. и Облезов А.С. составила настоящий акт о том, что использование результатов диссертационной работы Красновой И.А. при разработке и реализации проектов в ЗАО «ИнформИнвестИгрупп» позволяет решить следующие задачи:

— проводить классификацию трафика на основе индивидуальных статистических характеристик первых 15 пакетов потока с помощью алгоритма XGBoost в режиме реального времени;

— добавлять новые классы в существующую модель с помощью метода агломеративной кластеризации трафика на основе предварительно рассчитанной матрицы расстояний методом Extremely Randomized Trees;

Разработанный И.А. Красновой метод динамической классификации потоков позволяет более точно разделять IP – потоки и применять к ним методы управления трафиком для обеспечения качества обслуживания. Использование различных политик управления трафиком для классифицированных потоков позволяет более точно определять характеристики соглашения о качестве предоставляемых услуг (SLA).

**Технический директор** _____ **Вексельман М.И.**

**Советник Генерального директора** _____ **Зайцев Н.Л.**

**Инженер** _____ **Облезов А.С.**

Россия, 109240, Москва, ул. Верхняя Радищевская, д. 5, стр. 4
тел.: +7 (495) 787-1109, факс: +7 (495) 915-7950
www.iigroup.ru