МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 17

Под общей редакцией
чл.-корр. РАН, профессора Р.Л. Смелянского

МОСКВА – 2017

УДК 519.6+517.958
ББК 22.19
П75

Печатается по решению
Редакционно-издательского совета факультета
вычислительной математики и кибернетики МГУ имени М.В. Ломоносова

П75 **Программные системы и инструменты:** Тематический сборник/ Под ред. Смелянского Р.Л. – М: Издательский отдел факультета ВМиК МГУ (лицензия ИД №05899 от 24.09.2001 г.); МАКС Пресс, 2017. – № 17. – 200 с.

ISBN 978-5-89407-579-2
ISBN 978-5-317-05715-2

Данный выпуск сборника составлен по материалам работ студентов, получивших рекомендации научных семинаров кафедры Автоматизации Систем Вычислительных Комплексов, которую Л.Н. Королев создал и бессменно руководил, а также государственных аттестационных комиссий выпускников бакалавриата и магистратуры. Редколлегия сборника продолжает традицию издания сборника в память об этом выдающимся человеке. В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные проблемам создания информационно вычислительной инфраструктуры для научных исследований, проблемам современных компьютерных сетей, методам и средствам организации и управления облачными вычислениями, инструментальным средствам, обеспечивающим работу систем управления в реальном времени (СРВ), методам идентификации пользователей и анализу их поведения в компьютерных сетях в интересах информационной безопасности.

Статьи сборника будут интересны студентам, аспирантам и специалистам в области разработки прикладных программных систем с использованием новых информационных технологий.

Ключевые слова: информационно-телекоммуникационные технологии, программно-конфигурируемые сети, OpenFlow-коммутатор, централизованный контроллер, беспроводные Wi-Fi сети, контроль доступа приложений, компьютерная атака, облачные вычисления, центр обработки данных, алгоритм имитации отжига, виртуальные ресурсы, виртуализация сетевых функций, облачная платформа, задача о рюкзаке, система реального времени, интегрированная модульная авионика, построение расписаний, клоны исходного кода, средства разработки программ, информационная безопасность, свёрточные нейронные сети, задача аутентификации пользователей, гибридные методы распознавания пользователей, архитектура вычислительных систем, программирование..

УДК 519.6+517.958
ББК 22.19

ISBN 978-5-89407-579-2
ISBN 978-5-317-05715-2

СОДЕРЖАНИЕ

СБОРНИК

«Программные системы и инструменты»

От редколлегии:

Тематический сборник “Программные системы и инструменты” был инициирован Львом Николаевичем Королевым в 2000 году, как площадка где студенты и молодые ученые могли бы публиковать свои достижения. Все эти годы Лев Николаевич неустанно вел его, отбирал публикации, редактировал. Редколлегия сборника продолжает эту традицию в память об этом выдающимся человеке.

Этот выпуск сборника составлен по материалам работ студентов, получивших рекомендации научных семинаров кафедры Автоматизации Систем Вычислительных Комплексов, которую Л.Н. Королев создал и бессменно руководил, а также государственных аттестационных комиссий выпускников бакалавриата и магистратуры. В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные проблемам создания информационно вычислительной инфраструктуры для научных исследований, проблемам современных компьютерных сетей, методам и средствам организации и управления облачными вычислениями, инструментальным средствам, обеспечивающим работу систем управления в реальном времени (СРВ), методам идентификации пользователей и анализу их поведения в компьютерных сетях в интересах информационной безопасности.

В разделе «Из наследия Л.Н. Королева», в котором планируется публикация неопубликованных работ Льва Николаевича. В этом сборнике представлена статья «Архитектура и программирование», которая так и не была им опубликована.

Редколлегия

Смелянский Р.Л.

ИКТ инфраструктура нового поколения и задачи научно-образовательной сферы

Введение

Начало XXI века ознаменовалось осознанием следующих новых трендов в области информационно телекоммуникационных технологий (ИКТ):

- Рост трафика

Каждый из пользователей глобальной сети генерирует больше трафика, чем вся Всемирная паутина 30 лет назад. В 2014 году интернет-трафик вырос, по сравнению с 1984 годом, в 2,7 миллиардов раз (Cisco).

- Рост требований к качеству и разнообразию сетевых сервисов со стороны приложений, рост числа мобильных устройств

Число мобильных пользователей с 4,1 млрд в 2013 г. увеличилось до 4,9 млрд в 2018 г. В 2018 г. число мобильных устройств и подключений достигнет 10 миллиардов (8 миллиардов персональных мобильных устройств и 2 миллиарда межмашинных соединений). Для сравнения в 2013 г. общее число мобильных устройств и межмашинных соединений составило 7 миллиардов (Cisco).

- Смена парадигмы организации вычислений от клиент-серверной к ЦОД–облачной инфраструктуре

В 2005 году лишь около 5% серверов были виртуализированы. В 2014 года уже более 70% и большая их часть сосредоточена в ЦОДах (Gartner).

- Нарастание числа угроз и соответствующих атак их реализация

В 2015 году общее число инцидентов в области информационной безопасности возросло на 48% (!) по сравнению с 2013 г., составив в общей сложности 42,8 млн инцидентов. Это означает, что в среднем каждый день совершалось 117 339 кибератак. Если анализировать ситуацию за более длительный период, то, с 2009 г. совокупный среднегодовой темп роста (CAGR) числа выявленных инцидентов информационной безопасности ежегодно увеличивался на 66%. Причем, эти цифры нельзя назвать окончательными: за ними скрывается лишь общее число выявленных и подтвержденных инцидентов (PwC).

Эти тренды требуют пересмотра традиционной архитектуры Интернета, которая позволит:

- сократить затраты на эксплуатацию сетей за счет повышения уровня автоматизации;
- сократить сроки ввода в эксплуатацию новых сетевых сервисов, научиться управлять ими и виртуализировать;
- эффективно создавать виртуальную информационную инфраструктуру вне зависимости от ¬географического размещения физических ресурсов и пользователей;
- защитить информационную безопасность страны и обеспечитьсуверенитет национального территориального пространства.

Обозначенные выше тренды и связанные с ними примеры формируют новые вызовы и ставят новые задачи для научно-образовательной сферы России.

Исследования в современных условиях требуют развитой информационно-вычислительной инфраструктуры. Научно-образовательная среда так же сформировала свои требования к ИКТ инфраструктуре, необходимой для проведения НИОКР. Особую важность эти требования получили в связи с тем, что сегодня острую актуальность получают междисциплинарные проекты. Для таких университетов инфраструктура должна объединять воедино территориально распределённые ресурсы: модели, алгоритмы, коллекции экспериментальных данных, ИТ-ресурсы, набор исследовательских и технологических сервисов, взаимодействующих и согласованно управляемых в режиме реального времени.

Очевидно, что построение такой инфраструктуры под каждый научно-исследовательский проект является бесперспективным направлением по многим причинам. К началу XXI века было хорошо осознано, что компьютерные сети с традиционной TCP/IP архитектурой не позволяют решить эту проблему.

Уже в 2015 году российский телеком и промышленность в лице ГК «Ростехнологии» начали рассматривать планы на 2016 год по развитию пилотных проектов на основе технологий программно-конфигурируемых сетей (ПКС/SDN) и виртуализации сетевых сервисов (ВСС/NFV). В 2017 году планируются пилотные проекты по отработке миграции отечественных сетей на новые технологии. Таким образом, через 2 года ожидается массовый переход отечественного телекома на новые технологии. Однако сегодня ни один российский университет не ведет обучение по этим технологиям. На рынке труда сегодня наблюдается нехватка специалистов по компьютерным сетям, студенты не получают необходимых знаний, навыков и умения использовать облачные технологии.

1. Исследовательские проекты в области компьютерных сетей нового поколения в мире

На рисунке 1 приведены примерные объёмы исследовательских проектов в области компьютерных сетей нового поколения в мире. Рассмотрим подробней о состоянии этих проектов в ключевых странах мира.

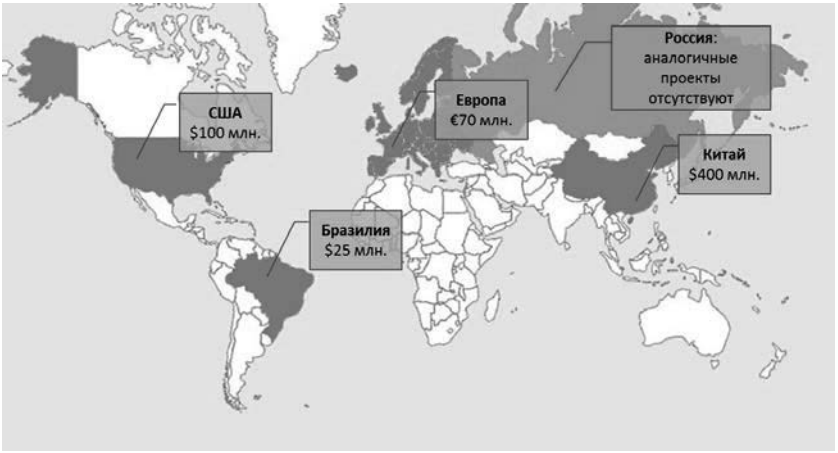

Рисунок 1. *Примерные объёмы исследовательских проектов в области компьютерных сетей нового поколения в мире.*

1.1 Соединенные Штаты Америки

В США поиски подходов к решению этой проблемы были начаты вначале 2000-х годов с программы PlanetLab, целью которого было создание инфраструктуры для разработки и тестирования новых протоколов. В 2006 году в Стэнфорде стартовал большой междисциплинарный проект CleanState. Его целью было рассмотреть с «чистого листа» требования к современной ИКТ-инфраструктуре и возможные пути их удовлетворения. Этот проект закончился в 2012 году. Его продолжением стал проект GENI (GlobalEnvironmentfor Network Innovations), который инициировал Национальный научный фонд США (NSF) с бюджетом более $ 100 млн. Сегодня в проекте участвует более 200 университетов США. Управляет проектом компания BBN Technology. Проект GENI являет собой прообраз Интернета нового поколения, это виртуальная лаборатория для проведения экспериментов в национальном масштабе.

В рамках проекта GENI фирма Ciena построила оптическую сеть (OTN), соединяющую ЦОДы на скорости 100Gbs. Эта сеть ЦОДов является испытательным полигоном для отработки методов и

построения соответствующих средств виртуализации, управления доступом, обеспечения информационной безопасности, проведения научных междисциплинарных исследований в разных прикладных областях.

1.2 Европа

В 2010 году в Евросоюзе стартовал проект Ophelia в рамках 7-ой рамочной программы по развитию приоритетных областей науки с бюджетом € 6 млн.

В 2012 запустился масштабный проект FED4FIRE (Future Internet ResearchEnvironment) – исследовательский проект в области компьютерных сетей и Интернета нового поколения. В проект входят университеты Бельгии, Великобритании, Франции, Германии, Австралии, Испании, Греции, Кореи. В странах Евросоюза так же начата подготовка специалистов для разработки программного обеспечения и эксплуатации сетей нового поколения.

1.3 Япония

В Японии создан проект AKARI – поиск инновационной архитектуры для Интернета, объединяет Национальный университет информационных и телекоммуникационных технологий (координатор), университеты Токио, Осаки, Кэйо, и технологический институт Токио.

1.4 Китайская Народная Республика

В 2015 году в КНР запущена программа Sea-Cloud Innovation Environment Update с бюджетом в $400 млн. Головным является Институт InformationalEngineering Китайской академии наук (лидер – профессор Ge Jingguo), Университет Циньхуа (под руководством профессора Jun Bi), а также Пекинский Университет (под руководством профессора Hong Mei).

Проект нацелен на отработку методов и средств виртуализации вычислительной и информационной инфраструктуры на основе федерализации ресурсов с использованием технологий SDN&NFV. Для этого строят сеть центров обработки данных (ЦОД) с небольшим количество серверов (10-20 штук) в каждом. ЦОДы объединяют высокоскоростными телекоммуникационными каналами.

1.5 Бразилия

В начале 2011 года создан проект FIBRE – Brazilian Testbed for Future Internet. Его задачей было создание единого пространства между Евросоюзом и Бразилий для совместных экспериментальных исследований в области Интернета и сетевой инфраструктуры. В FIBRE-EU вошли такие страны, как Испания, Италия, Франция,

Великобритания, Греция, Австралия; а FIBRE-BR объединила 10 ключевых университетов страны. На сегодняшний день в Правительство Бразилии вложило в проект порядка $25 млн.

1.6 Россия

В Российской Федерации исследования технологий Программно-Конфигурируемых Сетей (SDN) и Виртуализации Сетевых Сервисов (ВСС) были начаты в МГУ им. Ломоносова в 2011 году, а с 2012 эти исследования были сконцентрированы в Центре Прикладных Исследований Компьютерных Сетей – некоммерческой организации, которой Фонд «Сколково» выделил грант на 5 лет. С 2013 года НП «ЦПИКС» совместно с МГУ безуспешно пытаются инициировать проект для решения указанной выше проблемы в России. Суть инициативы GRANIT (Global Russian Advanced Network Initiative) изложена ниже.

2. Проект GRANIT

Основная цель предлагаемого проекта GRANIT – разработка средств построения виртуальной информационно-вычислительной инфраструктуры на ресурсах географически распределенной сети Центров Обработки Данных (ЦОД) на основе технологии ПКС и инструментальная поддержка профильной деятельности специалистов в соответствующей предметной области с целью существенного повышения эффективности этой деятельности.

Задачи проекта GRANIT заключается в «погружении» всех заинтересованных территориально-распределенных групп специалистов, работающих над соответствующей проблемой, в защищенное проблемно-ориентированное облако.

В ходе реализации Проекта предполагается создать:

- *Программную платформу*, которая обеспечивает базовые функции по сбору информации о состоянии ресурсов сети ЦОД'ов, выполнению управляющих приложений, передаче управляющих воздействий на сетевые коммутаторы; набор управляющих приложений, отвечающих за маршрутизацию, балансировку нагрузки, оптимизацию распределения ресурсов, соблюдение политик безопасности, обеспечение заданного уровня качества сервиса, стыковку с сетями традиционной архитектуры; логически централизованную управляющую платформу, которая обеспечивает согласованное видение состояния ресурсов сети ЦОД'ов как для приложений на одном узле (площадке), так на нескольких географически распределённых узлах.
- *Средства построения виртуальной информационной инфраструктуры*, включающей вычислительные, сетевые ресурсы и

хранилища данных, для проведения исследований в разнообразных предметных областях знаний. Достижение результатов Проекта даст возможность замещения импортного сетевого оборудования (более 90% оборудования сегодня - зарубежное). Вместо дорогих проприетарных маршрутизаторов в управлении сетью будут использоваться обычные серверы широкого применения с отечественным ПО.

- *Экспериментальную сеть микро ЦОД'ов*, состоящую из географически распределённых площадок, объединённых национальными высокоскоростными каналами связи. На этих площадках будут установлены экспериментальные сетевые сегменты. Площадки на первом этапе (14 точек) планируется разместить в организациях, участвующих в решении задач Проекта, а впоследствии и в организациях – пользователях экспериментальной инфраструктуры (ещё около 20 площадок). На площадке размещается унифицированный комплект оборудования: кластер из 10 серверов, состоящий не менее чем из 16 вычислительных узлов каждый, количество ядер процессора не менее 8 для поддержки не менее 256 виртуальных машин, сетевой коммутатор, поддерживающий стандарт управления OpenFlow 1.3 и выше и суммарной производительностью сетевых интерфейсов на каждый вычислительный узел не менее 10 Gbs, магистральные каналы связи.

Основные тематики предполагаемых исследований в проекте GRANIT включают в себя:

- Разработку специализированных алгоритмов управления и виртуализации вычислительных, сетевых ресурсов и хранилищ данных, включая виртуализацию топологии сети.
- Разработку специализированного языка описания виртуальной информационной инфраструктуры.
- Разработку среды автоматического прогона экспериментов, на основе описания полного процесса подготовки и проведения эксперимента. Разработка специализированного языка описания экспериментов.
- Подготовку SaaS-портала для экспериментаторов из различных областей наук. На начальном этапе предполагается создание базы шаблонных виртуальных машин с предустановленными различными системами математического и имитационного моделирования. Предполагается развитие проекта как 'BringYourOwnImage' (BYOI) testbed.
- Разработку специализированного портала для экспериментаторов и разработчиков проекта. С возможностью ролевого доступа к ресурсам системы, а также

интеллектуального доступа к результатам уже состоявшихся экспериментов.

Ожидаемые результаты проекта GRANIT:

- Замещение импортного сетевого оборудования: вместо дорогих проприетарных маршрутизаторов в управлении сетью будут использоваться обычные серверы широкого применения с отечественным ПО.
- Рост рынка российского программного обеспечения, в котором российские разработчики благодаря сильной математической школе – традиционно сильны. ПО - наиболее динамичный сегмент мирового рынка ИТ, ежегодный рост которого в последние несколько лет превышал 6%. По оценкам аналитиков, на нем порядка 80% всех продаж приходится на иностранные продукты. Между тем участники рынка говорят, что российскому софту вполне по силам занимать не нынешние 20%, а 80% рынка.
- Появится новый сегмент рынка ПО – управляющие приложения для сетевых контроллеров. Его развитие будет способствовать расширению числа сетевых услуг. Аналитические исследования показывают, что уже в 2016 году этот рынок достиг $1,8 млрд.
- Формирование отечественной школы исследователей в области компьютерных сетей, готовые кадры для эксплуатации сетей. Подключение к ЦКП организаций с уникальными научными ресурсами позволит повысить мобильность специалистов за счет виртуализации инфраструктуры информационного пространства исследователя.

3. Заключение

Актуальность рассматриваемой проблематики для Российской Федерации определяется отсутствием отечественных средств построения сетей, управления сетями и специализированными сервисами в них. Для текущей ситуации проведения НИР в России характерно наличие разрозненных ресурсных центров (что существенным образом сказывается на эффективности инвестиций). Все более существенную роль в современных исследовательских проектах играет мобильность учёных.

Аграновский М.Л, Шалимов А.В.

Применение хеш-функций для ускорения работы программных OpenFlow-коммутаторов

Введение

Основная идея *программно-конфигурируемых сетей (Software-Defined Networking, SDN)* – это разделение уровней управления и передачи данных в сетевых устройствах. Уровень управления выносится в отдельное централизованное приложение – контроллер, работающее на отдельном вычислительном сервере. Уровень передачи данных остается на сетевых устройствах и связывается с уровнем управления специализированным сетевым протоколом управления. Наиболее распространенным таким протоколом является OpenFlow [1]. Парадигма SDN постулирует наличие единого уровня управления сетью коммутаторов, упрощая тем задачи управления и мониторинга сети. Открытый протокол управления и стандартизация архитектуры коммутаторов позволяют строить компьютерные сети независимо от производителей сетевого оборудования. Появление инструментариев автоматизированного конфигурирования уровня данных повышает гибкость и универсальность сетевого оборудования [2].

Ранее в высокопроизводительных компьютерных сетях применялись в основном аппаратные решения на *интегральных схемах специального назначения (Application Specific Integrated Circuit, ASIC)* или *сетевых процессорах (Network Processor Unit, NPU)*. Снижение требований к аппаратуре сделало возможным применение *программных коммутаторов на базе x86-серверов*. Данная работа посвящена работе с ними.

Программные коммутаторы – это программы, предназначенные для запуска на вычислительных серверах архитектуры x86 и обеспечивающие коммутацию пакетов в системах передачи данных. Использование серверов общего назначения означает отсутствие специфичных физических модулей, применяемых в аппаратных коммутаторах для повышения производительности:

- модулей разгрузки центрального процессора (ASIC, NPU), на которых в аппаратных устройствах выполняется бóльшая часть уровня передачи данных,
- модулей памяти с постоянным временем поиска (TCAM, CAM).

Важность применения программных коммутаторов вызвана тем, что они обладают наибольшими функциональностью и гибкостью, допускают удаленное развертывание в рамках платформ облачных вычислений. Только на них сейчас возможна полная поддержка стандарта OpenFlow. Однако программные коммутаторы уступают аппаратным решениям по производительности.

Данная статья посвящена исследованию использования аппарата хэш-функций для повышения скорости работы программных OpenFlow-коммутаторов. В статье предложен метод повышения производительности коммутатора за счет сокращения суммарной длины полей заголовков сетевых пакетов, по которым вычисляется хэш-функция (см. раздел 3).

Изложение структурировано следующим образом. В разделе 1 описывается метод кеширования часто исполняемых OpenFlow-правил в хэш-таблицы, применяемый в промышленности для повышения производительности программных OpenFlow-коммутаторов. У систем, использующих этот метод, выделяются общие недостатки, негативно влияющие на производительность коммутаторов. В разделе 2 вводится разновидность такой системы, лишенная ряда недостатков. В разделе 3 для нее предлагается способ повышения производительности: вместо вычисления хеш-функции по всему заголовку пакета предлагается вычислять ее лишь по тем полям, по которым в данной таблице потоков выполняется сравнение. Это становится возможно благодаря учету специфики приложений, работающих в таблицах потоков OpenFlow-коммутатора. Раздел 4 содержит описание проведённого экспериментального исследования предложенной модификации. В разделе 5 делаются выводы по результатам выполненной работы.

1 Система кеширования правил в программных OpenFlow-коммутаторах

Основа модели коммутатора OpenFlow – таблицы потоков. **Потоком** в терминологии OpenFlow называется множество пакетов, заголовки которых удовлетворяют некоторому предопределенному шаблону. Для каждого потока определяется набор ***действий***, которые коммутатору предписывается выполнять над всеми пакетами данного потока. Такие пары $(шаблон\_сравнения, набор\_действий)$ хранятся в таблицах, называемых **таблицами потоков**. Последовательность таких таблиц образует ***конвейер***, через который следуют все поступающие на коммутатор пакеты. В программных коммутаторах таблицы потоков реализуются в виде деревьев поиска. **Достоинство** деревьев поиска – гибкость реализации, в результате чего возможны префиксный и тернарный виды поиска (например, Patricia tree

в коммутаторе Lagopus [5]). Данная возможность позволяет задавать шаблон сравнения посредством маски, что эффективнее явного задания набора значений полей сравнения. **Недостаток** деревьев поиска – низкая производительность алгоритмов поиска, недостаточная для работы в сетях операторов связи.

Для повышения производительности коммутатора создается система кеширования наиболее часто выполняемых правил в хеш-таблицы. Для каждого поступающего на коммутатор пакета по значениям предопределенного множества полей заголовка вычисляется хеш-функция. По полученному значению хеш-функции в хэш-таблицу записывается последовательность действий, которая будет выполняться над каждым пакетом с соответствующим заголовком, или одно действие, результирующее эту последовательность [3]. Под **результирующим действием** понимается следующее. Пусть над пакетами некоторого потока предписано выполнять следующие действия (посредством инструкций ApplyAction или WriteAction):

- изменять mac-адрес источника на mac-адрес коммутатора: $eth\_src = AA:BB:CC:DD:EE:FF$,
- заменять первые 12 бит mac-адреса на номер vlan-метки: $eth\_src = vlan\_tag \,||\, (eth\_src << 12) >> 12)$.

Тогда в хэш-таблицу будет записано действие, объединяющие вышеуказанные:

$$eth\_src = vlan\_tag \,||\, 00:0B:CC:DD:EE:FF.$$

Система кеширования называется «быстрым путем» (fast path) уровня данных программного коммутатора. Она противопоставляется «медленному пути» (slow path) – полноценному конвейеру OpenFlow. Управление на slow path передается только в случае кеш-промаха в fast path.

В рамках данного исследования рассмотрены системы кеширования коммутаторов Open vSwitch [3] и Lagopus [4], как наиболее популярных решений с открытым исходным кодом, предоставляющих наиболее полную поддержку OpenFlow [6]**.** В них создается одна хеш-таблица, в которой для каждого пакета запоминаются список указателей на необходимые к выполнению действия (Lagopus) или результирующее действие всего конвейера коммутатора по всем таблицам потоков (Open vSwitch) (см. *Рисунок 1*). Подход с одной таблицей имеет следующие недостатки:

1. **Ограничение производительности fast path.**
   1.1. **Высота хэш-таблицы (число записей).** При отображении n таблиц потоков в одну в худшем случае применяется декартово произведение результирующих правил каждой таблицы. Необходимость хранить в одной таблице большое число записей повышает вероятность коллизий и

вынуждает повышать длину хэша, что негативно сказывается на времени вычисления и сравнения хэш-значений.

1.2. **Размер памяти, занимаемый хэш-таблицей**. Большая таблица не помещается в страницы виртуальной памяти, следовательно, происходят частые промахи TLB-кеша процессора, что понижает скорость работы коммутатора.

1.3. **Блокировки хэш-таблицы.** Чтобы ограничивать рост высоты таблицы, Open vSwitch удаляет редко используемые строки. При операции удаления происходит блокировка таблицы, а значит – всего fast path. Синхронизация статистических счетчиков между slow path и fast path также приводит к блокировкам, что понижает производительность.

2. **Накладные расходы в slow path.** Необходимость собирать «трассу выполнения» конвейера дополнительно замедляет работу конвейера коммутатора. В дополнение к этому, в Open vSwitch также вычисляется результирующее правило.

3. **Сложность программной реализации.** Задачи сбора «трассы выполнения» конвейера и сведения ее к результирующим действиям повышают сложность архитектуры продукта.

В данной работе рассматривается подход, позволяющий снизить влияние указанных недостатков на производительность коммутатора.

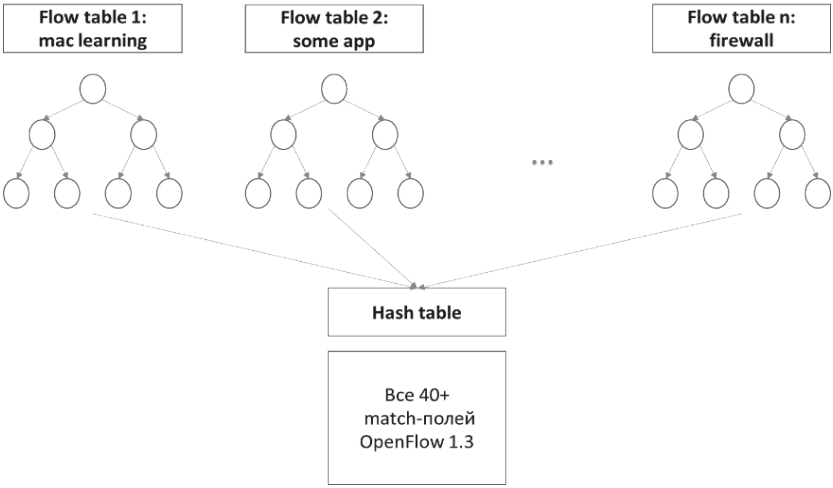

Рисунок 1. *Система кеширования с одной хэш-таблицей*

2 Система кеширования с множественными хэш-таблицами

Для преодоления ограничений производительности fast path с одной хэш-таблицей (см. *Раздел 1* выше) предлагается создавать в системе кеширования несколько таблиц: по одной для каждой таблицы потоков (см. *Рисунок 2*). Так у каждой таблицы потоков появляется свой кеш. В результате на каждом этапе OpenFlow-конвейера поиск сначала производится в ассоциированным с данным этапом кеше, и только в случае кеш-промаха производится поиск в таблице потоков.

Получаемые хэш-таблицы независимы и имеют относительно небольшой размер (в сравнении с одной общей), что положительно сказывается на производительности: пропадает необходимость хранить декартово произведение правил, снижаются частоты TLB-промахов и блокировок, становится возможным использование хэша меньшей длины. В предлагаемой системе создание результирующих правил может быть опущено, т.к. его полезный вклад уменьшится в виду распределения множества ранее объединяемых правил по отдельным таблицам.

Однако, в отличие от подхода с одной таблицей, возникает необходимость вычисления хэш-функции при входе пакета в каждую хэш-таблицу, что отрицательно сказывается на производительности коммутатора:

- Пусть пакет движется по конвейеру из n таблиц, и в таблице $m < n$ были изменены значения некоторых полей заголовка пакета.
- Пусть в таблице $k \in [m+1, n]$ выполняется сравнение по некоторому из измененных в таблице m полей.
- Тогда, чтобы над пакетом выполнились наборы действий, соответствующие измененным значениям полей заголовка пакета, необходимо заново вычислить хэш от заголовка этого пакета.

Встает задача снижения временных затрат на вычисления хэша от заголовков сетевых пакетов.

3 Сокращение времени вычисления хэша от заголовков сетевых пакетов за счет учета специфики OpenFlow-приложений, работающих с таблицами потоков.

В модели OpenFlow требуемый от компьютерной сети функционал реализуется ***приложениями***, работающими на контроллере. Этот функционал достигается за счет того, что приложения записывают ***правила*** в таблицы потоков.

В данной работе предлагается метод повышения производительности коммутатора за счет сокращения времени вычисления хэша от заголовков сетевых пакетов. Он основан на следующих соображениях. Как правило, с одной таблицей потоков работает несколько приложений, реализующих требуемый для сети функционал. Как правило, одно приложение производит сравнение по небольшому подмножеству множества всех доступных в OpenFlow полей сравнений. Значит, в большинстве таблиц потоков производится сравнение по небольшому подмножеству всех полей сравнений. Это приводит к сокращению в разы числа полей заголовков, которые необходимо учитывать при вычислении значения хеш-функции. Для этого для каждой таблицы потоков дополнительно необходимо сохранять набор полей заголовков, по которым в ней может производиться сравнение, и вычислять в соответствующей хэш-таблице хеш-функцию только от этих полей. Таким образом производительность коммутатора повысится за счет сокращения времени вычисления хеша и за счет уменьшения длины ключа сравнения (в хеш-таблице для предотвращения коллизий при совпадении хэшей производится сравнение исходных битовых слов).

С учетом высказанных соображений можно организовать работу коммутатора по следующему алгоритму:

1. **Инициализация хеш-таблиц.** На этапе конфигурирования коммутатора параметры хеш-таблиц настраиваются в соответствии с множествами полей сравнения, используемых в таблицах потоков.
2. **Прием пакетов.** Поступающие на коммутатор сетевые пакеты обрабатываются им следующим образом:
   2.1. **Выбор полей заголовка по маске данной хэш-таблицы.** На входе в каждую таблицу конвейера из заголовка извлекаются и склеиваются в одно битовое слово поля, указанные в конфигурации таблицы на шаге 1 (см. *Рисунок 3*).

2.2. **Вычисления хэша.** Полученное битовое слово передается в указанную в конфигурации данной таблицы хэш-функцию для вычисления хэша.

2.3. **Поиск в хэш-таблице.** По хэш-значению производится поиск в хэш-таблице, ассоциированной с данной таблицей потоков.

2.3.1. Если правило, отвечающее данному заголовку пакета, найдено, то оно исполняется (fast path).

2.3.2. В противном случае управление передается на полноценную обработку в таблицу потоков (slow path). По завершении обработки, таблица потоков возвращает результирующее действие, которое записывается в хеш-таблицу по ранее посчитанному хэшу.

2.3.3. В зависимости от исполненного правила, управление передается на впереди стоящий этап конвейера (тогда алгоритм повторяется с шага 2.1), либо конвейер завершается.

3. Завершение работы.

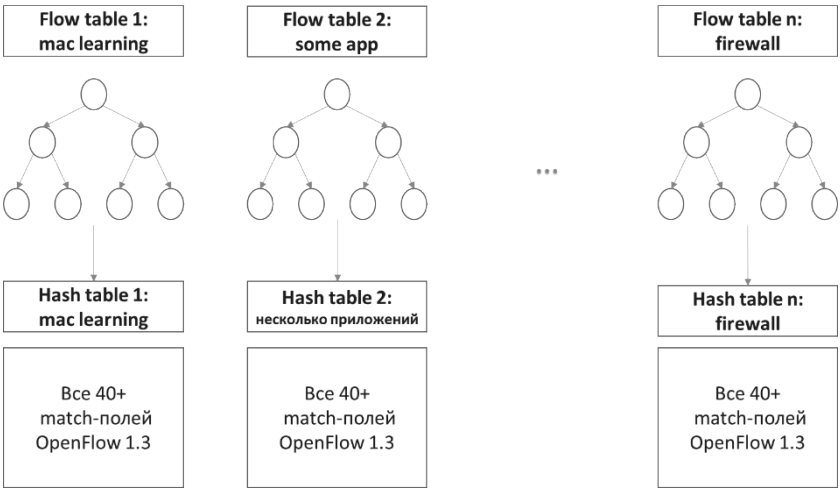

Рисунок 2. *Система кеширования с отдельной хеш-таблицей для каждой таблицы потоков*

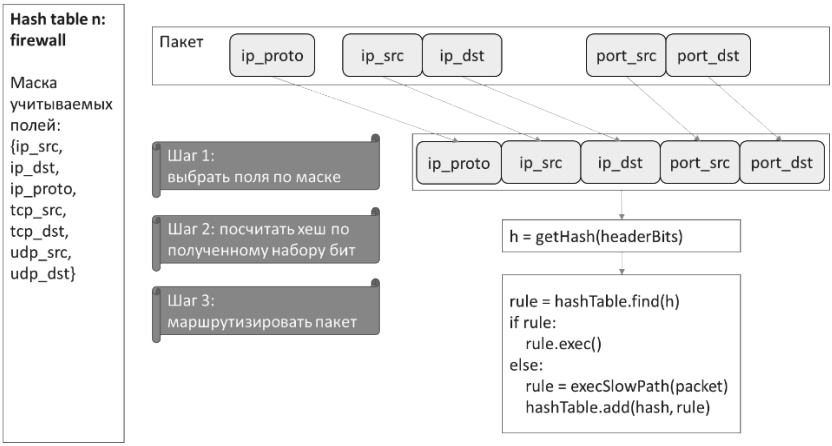

Рисунок 3. *Список действий, выполняемых для каждой таблицы потоков*

4 Экспериментальное исследование

4.1 Алгоритм работы модельного приложения

Предложенная оптимизация системы кеширования по времени работы экспериментально исследована на сетевом приложении, моделирующем работу программного OpenFlow-коммутатора с двумя системами хеширования: с учетом специфики приложений и без. При поступлении на коммутатор очередного пакета поочередно выполнялись поиски в обеих системах кеширования и аккумулировалось число тактов процессора, потраченных на каждый из поисков. По окончанию работы коммутатора вычислялся относительный прирост производительности, вызванный применением предложенной системы кеширования:

$$increace = 1 - \frac{ticksOnOptimizedFastPath}{ticksOnUsualFastPath}.$$

4.2 Методика экспериментальной оценки производительности

В рамках исследования проведена имитация работы 4 сетевых приложений, показывающих использование различного количества полей сравнения (см. *Таблица 1*).

Схема стенда представлена на *Рисунок 4*. Модельное приложение запускалось на вычислительном сервере (Intel Xeon E3-1240 V2 @ 3.40GHz, 8 ядер, 2 ГБ ОП, Ubuntu 14.04 64-bit). Порт сетевой карты (Intel 82599 10 Gigabit TN) изымался из-под управления сетевого стека

Linux и передавался приложению в эксклюзивное пользование посредством инструментария DPDK. Данный порт был соединен с портом данной сетевой карты, с которого подавалась нагрузка трафик-генератором pktgen [5] в соответствии со следующей конфигурацией: 50 серий по 100 000 пакетов размера от 300 до 400 байт.

4.3 Результаты экспериментального исследования

Результаты исследования (см. *Рисунок 5*) подтверждают пропорциональность отношения суммы длин используемых полей сравнений $field_i$ к сумме длин всех доступных в OpenFlow полей сравнений к отношению временем работы оптимизированного fast path коммутатора $ticks\_optimized$ ко времени работы оригинального $ticks\_usual$: $\frac{\sum_{i=1}^{n} len(field_i)}{\sum_{i=1}^{OF\_NFIELDS} len(field_i)} \sim \frac{ticks\_optimized}{ticks\_usual}$. Тестирование показало средний прирост производительности системы кеширования в **2.4%**. Прирост производительности доказывает применимость предложенного метода.

Имя приложения	**Описание**	**Суммарная длина сравниваемых полей заголовков, Байт**
vlan-based-app	Приложение, основанное на анализе VLAN-меток	2 // vlan_id
mac-learning	Приложение, реализующее L2-коммутатор	12 // eth_dst, eth_src
NAT	Сервис трансляции сетевых адресов (NAT)	18 // 5tuple: ip_src, ip_dst, tcp_src, tcp_dst, udp_src, udp_dst, ip_proto
synthetic	Модельное приложение, использующее 1/3 всех доступных в OpenFlow 1.3 полей сравнения	53 // KEY_LEN_OF13 / 3

Таблица 1. *Тестовые приложения*

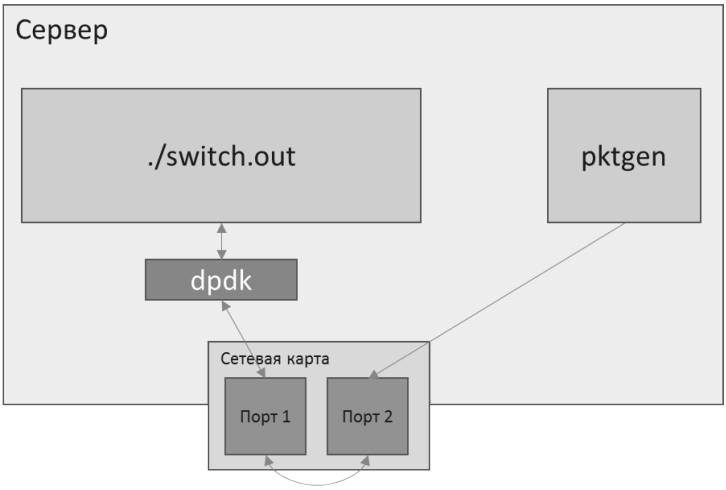

Рисунок 4. *Схема стенда*

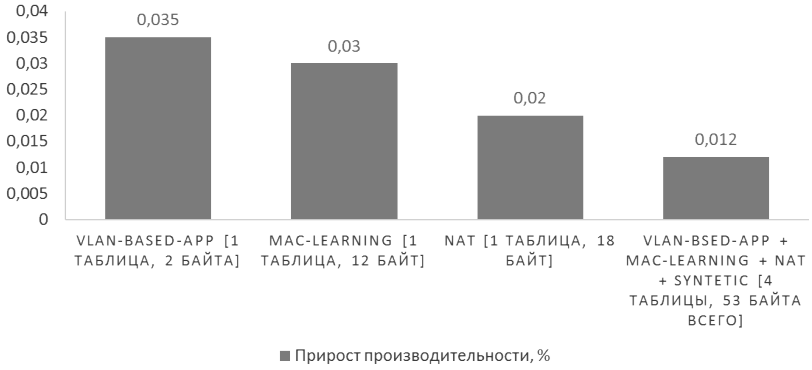

Рисунок 5. *Относительный прирост производительности системы кеширования коммутатора на наборах тестовых приложений.*

5 Заключение

В работе для программных OpenFlow-коммутаторов предложен метод организации системы кеширования наиболее часто выполняемых OpenFlow-правил. Он основан на создании отдельной хеш-таблицы для каждой таблицы потоков и решает ряд ограничений производительности систем с одной таблицей. Для него предложена модификация, позволяющая уменьшить временя выполнения. Она основана на учете для каждой таблицы потоков множеств полей сравнений, используемых приложениями в данной таблице. Предложенная модификация позволяет повысить производительность данного метода за счет сокращения временных затрат на поиск в хеш-таблице. Применимость модификации исследована и доказана на сетевом приложении, моделирующем работу программного OpenFlow-коммутатора. Задача конфигурирования может производиться OpenFlow-контроллером автоматический при помощи технологий P4 [2] / POF [8].

Литература

1. McKeown N. et al. OpenFlow: enabling innovation in campus networks //ACM SIGCOMM Computer Communication Review. – 2008. – Т. 38. – №. 2. – С. 69-74.
2. Bosshart P. et al. P4: Programming protocol-independent packet processors //ACM SIGCOMM Computer Communication Review. – 2014. – Т. 44. – №. 3. – С. 87-95.
3. Shelly N. et al. Flow caching for high entropy packet fields //ACM SIGCOMM Computer Communication Review. – 2015. – Т. 44. – №. 4. – С. 151-156.
4. Pfaff B. et al. The Design and Implementation of Open vSwitch //NSDI. – 2015. – С. 117-130.
5. NTT Communications. Документация программного OpenFlow-коммутатора Lagopus. -- URL: http://www.lagopus.org/lagopus-book/en/html/ (Дата обращения: 17.05.2017)
6. Ryu SDN Framework Community. Результаты сертификации сетевых коммутаторов по наличию поддержки протокола OpenFlow версий 1.3 и 1.4. – URL: https://osrg.github.io/ryu/certification.html (Дата обращения: 18.05.2017)
7. Olsson R. Pktgen the linux packet generator //Proceedings of the Linux Symposium, Ottawa, Canada. – 2005. – Т. 2. – С. 11-24.
8. Song H. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane //Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. – ACM, 2013. – С. 127-132.

Пашков В.Н., Гуськов Д.А.

Метод выбора отказоустойчивого размещения контроллеров в глобальных программно-конфигурируемых сетях

Введение

В программно-конфигурируемых сетях (ПКС) [1] функции управления сетью вынесены из сетевых устройств (маршрутизаторов) в специальное программное обеспечение, называемое контроллером или сетевой операционной системой. ПКС позволяет централизовать управление сетевыми устройствами, потоками данных и сетевыми сервисами, что в свою очередь даёт возможность гибкого управления и масштабирования сети.

Одним из наиболее развивающихся протоколов, обеспечивающих взаимодействие между коммутаторами и контроллером в ПКС, является протокол OpenFlow [2]. В данной статье под коммутаторами будем понимать коммутаторы, поддерживающие протокол OpenFlow.

OpenFlow коммутатор содержит одну или несколько таблиц потоков, поддерживают постоянное активное защищенное логическое соединение с удалённым контроллером. В таблицах потоков содержится список правил для обработки пакетов. Эти правила на коммутаторе устанавливаются контроллером. Коммутатор, получив новый пакет, осуществляет поиск в таблице потоков соответствующего правила. Если правило найдено, то над данным пакетом выполняются инструкции, содержащиеся в правиле. Если же подходящего правила не найдено, то коммутатор отсылает заголовок данного пакета (или весь пакет) контроллеру. Контроллер, получив данное сообщение, генерирует и добавляет в таблицу потоков коммутатора соответствующее правило для данного пакета. Совокупность пакетов с одинаковыми заголовками будем называть потоком. Таким образом, если для нового потока на коммутаторе не существует соответствующего правила, то это правило будет установлено контроллером для всего потока при появлении на коммутаторе первого пакета данного потока. Временем установления нового потока в сети будем называть время между приходом первого пакета нового потока на коммутатор и установкой соответствующих правил контроллером на всех коммутаторах сети, через которые будет проходить данный поток.

Несмотря на возможность гибкого управления сетевой инфраструктурой, ПКС обладает и следующим существенным

недостатком: централизация управления сетью в контроллере приводит к появлению единой точки отказа. Сбои и отказы в работе контроллера могут приводить к ошибкам в работе сети, или к полному прекращению ее функционирования [3]. Одним из решений данной проблемы является размещение в сети распределённого контроллера. Под распределённым контроллером будем понимать множество экземпляров логически централизованного контроллера, запущенных на различных серверах, а экземпляры контроллера будем называть просто контроллерами [4]. Для каждого контроллера определяется набор коммутаторов, которыми управляет данный контроллер. Если в результате отказа в сети некоторый коммутатор потерял соединение с управляющим контроллером, то любой другой контроллер, имеющий связь с данным коммутатором, может взять управление коммутатором на себя [5]. Под доменом контроллера будем понимать множество коммутаторов, у которых данный контроллер является управляющим. Будем считать, что все контроллеры поддерживают соединения со всеми коммутаторами сети, но управляют только коммутаторами из своего домена. Если новый поток проходит через несколько доменов в сети, то установление нового потока происходит следующим образом: коммутатор, не найдя подходящего правила в таблице, посылает соответствующее сообщение своему управляющему контроллеру. Контроллер определяет, что этот поток проходит и через другие домены сети и посылает запрос на синхронизацию другим контроллерам. Так как все контроллеры видят всю сеть, то все они строят одинаковый маршрут для данного потока. Далее каждый контроллер устанавливает соответствующие правила на коммутаторы своего домена, через которые проходит новый поток.

В статье рассматриваются глобальные компьютерные сети с архитектурой ПКС. Одной из особенностей WAN сетей является существенные задержки на физических каналах связи между коммутаторами, связанные с географической удаленностью точек их размещения. Задержки между коммутаторами WAN сети существенно влияют на задержки установления потоков, так как время на передачу управляющих сообщений между коммутатором и контроллером может в разы превышать время на обработку этих сообщений контроллером. Перегрузка контроллера может привести к значительному росту времени установления нового потока и некорректной работе сети. Чтобы избежать перегрузки контроллеров, нужно либо разместить больше контроллеров в сети и перераспределить нагрузку между ними, либо повышать производительность контроллеров. Таким образом, одной из актуальных и значимых проблем WAN сетей с архитектурой ПКС является задача размещения контроллеров, которая заключается в определении количества контроллеров, мест их размещения, а также распределения управления коммутаторами между контроллерами.

1. Постановка задачи

Дан граф $G = (V, E)$, описывающий магистральную ПКС сеть, где $V = \{v_i \mid i = \overline{1..n}\}$ – множество вершин графа G, соответствующее множеству коммутаторов, размещённых в узлах ПКС сети, а $E = \{e_i = (v_k, v_l) \mid v_k, v_l \in V\}$ – множество рёбер в графе, соответствующее множеству физических каналов связи между коммутаторами в сети. Для графа задано множество весов рёбер Lat, соответствующее задержке на физических каналах связи, и для каждого канала связи $e_i = (v_k, v_l)$ задана задержка передачи сигнала $l_{v_k, v_l} \in Lat$. Для каждого коммутатора v_i задано максимальное количество запросов на установление нового потока, которое он может генерировать за секунду $sl_v = load(v)$. Обозначим множество всех sl_v в сети как SL.

Обозначим за $C = \{c_i \mid i = \overline{1..k}\}$ – множество контроллеров распределённой платформы управления ПКС сети. Для каждого контроллера $c_i \in C$ задано максимальное количество запросов на установление нового потока, которое он может обработать за секунду $cl_c = load(c)$, а также стоимость обслуживания контроллера $cc_c = cost(c)$. Размещением контроллеров в сети будем называть множество $CP = \{cp_{v,c} \mid v = \overline{1..n}, c = \overline{1..k}\}$, где $cp_{v,c}$ принимает значение 1, если в узле $v \in V$ размещён контроллер $c \in C$, и 0 - в противоположном случае.

Определим распределение управления сетью как множество пар: $CD = \{\langle v_i, c_m \rangle \mid v_i \in V; c_m \in C\}$, где c_m является основным контроллером для коммутатора v_i. Для каждого управляющего соединения $mc_{i,j} = [v_i, c_j]$ коммутатора v_i с контроллером c_j задана стоимость обслуживания $scc_{i,j} = cost(mc_{i,j})$. Обозначим суммарную стоимость обеспечения управления сетью, как $SCost = \sum_{i=0}^{k} cc_i + \sum_{\forall i,j: \exists [v_i, c_j]} scc_{i,j}$.

Для сети задано максимально-допустимое значение задержки установления нового потока $Lmax$. Определим функцию задержки

сигнала, следующего по кратчайшему пути между узлами сети как $lat(v_k, v_l)$, где $v_k, v_l \in V \bigcup C$. Время синхронизации всех контроллеров при установлении нового потока зависит от задержки между наиболее удалёнными контроллерами и обозначается как $CST = syn\_time\left(\max_{c_i, c_j \in C} lat(c_i, c_j)\right)$. $CST = 0$, если новый поток проходит только через коммутаторы одного домена. Задержка NFL_{v_k, v_l} на установления нового потока, выходящего из узла v_k в узел v_l,где $v_k, v_l \in V$ складывается из задержки на передачу сигнала от коммутатора v_k до его управляющего контроллера, из времени синхронизации контроллеров CST и из задержки на установление соответствующих правил на все коммутаторы, через которые проходит данный поток.

Обозначим множество сценариев отказов a коммутаторов и b контроллеров как $G_a^b = \left\{ G_x^y \mid x = \{x_1, \ldots, x_a\},\ y = \{y_1, \ldots, y_b\} \right\}$, где G_x^y – граф, описывающий сеть, в которой отказали коммутаторы $v_{x_1}, \ldots, v_{x_a}$ и контроллеры $c_{y_1}, \ldots, c_{y_b}$.

Тогда требуется найти функцию размещение контроллеров:

$$F_{\text{разм. контр.}}\left(G = (N, E),\ Lat, \{cl_i\}, \{sl_i\},\ \{cc_i\},\ \{scc_{i,j}\},\ syn\_time,\ L_{max}\right) \to CP,\ CD$$

Чтобы:

- $SCost \to min$: суммарная стоимость обслуживания области управления должна быть минимальна,
- $avg_{\forall i,j}\left(NFL_{i,j}\right) \to min$: средняя задержка на установление нового потока должна быть минимальной,
- $\forall v_i \in V\ \exists! c_m \in C : \langle v_i, c_m \rangle \in CD$: для каждого коммутатора существует единственный управляющий контроллер.
- $\forall G \in G_0^0 \bigcup G_1^0 \bigcup G_0^1 \bigcup G_1^1,\ \forall i, j = \overline{1..n}\ NFL_{i,j} < Lmax$: при отказе не более одного контроллера и не более одного коммутатора задержка на установление нового потока не должна превышать максимально допустимую.
- $\forall G \in G_0^0 \bigcup G_1^0 \bigcup G_0^1 \bigcup G_1^1,\ \forall i = \overline{1..k} \sum_{j \mid \exists mc_{i,j}} sl_{v_j} < cl_i$: при отказе не более одного контроллера и не более одного комму-

татора загруженность каждого из контроллеров не превышает их производительность.

Данная задача сводится к основной задаче линейного программирования и является NP-трудной.

2. Обзор существующих подходов к решению задачи размещения контроллеров в ПКС

Впервые проблема размещения контроллеров для программно-конфигурируемых сетей была рассмотрена в работе [6]. Авторы ввели две метрики: задержка от коммутатора до управляющего контроллера в среднем и в худшем случае. Авторы исследовали влияние таких факторов, как требования к размещению, выбор метрики и топология сети на оптимальное решение. В результате исследований на топологиях из библиотеки Topology Zoo [11] было получено, что решение сильно зависит от самой топологии и в 10% топологий не стоит отдавать приоритет какой-либо из метрик и искать решение на их границе Парето.

В работах [7] и [8] для каждого физического соединения, каждого коммутатора и каждого контроллера определена его надёжность. Требуется разместить контроллеры в сети так, чтобы для каждого коммутатора вероятность потери соединения со всеми контроллерами не превышала заданного значения, а количество контроллеров было минимально.

В работе [9] авторы дополняют метрики, введённые в работе [8] четырьмя метриками:

- *P_max_latency(x)* – максимальная задержка между коммутатором и управляющим контроллером среди всех сценариев отказа *x* контроллеров,
- *P_controller-less(x)* – максимальное число бесконтроллерных коммутаторов среди всех сценариев отказа *x* контроллеров,
- *P_imbalance(x)* – максимальный дисбаланс среди всех сценариев отказа *x* контроллеров, где для каждого сценария дисбалансом называется максимальная разница в количестве управляемых коммутаторов среди всех контроллеров,
- *P_controller_latency* – максимальная задержка между контроллерами.

Авторы разработали фреймворк, который ищет решение задачи размещения контроллеров на границе Парето для вышеупомянутых метрик, однако алгоритм работы фрейморка не публикуется.

В статье [9] решается задача размещения контроллеров в ПКС масштаба WAN с учётом требования устойчивости сети к отказу не более чем одного контроллера и соединения между контроллерами, а

также определения резервного контроллера для каждого коммутатора. Автор использовал метрики из статьи [8] и ввёл ограничения на количество коммутаторов во всех доменах сети. Размещение контроллеров производится с помощью разбиения сети на области двусвязности и применением методов кластеризации *k-means* или k-medians для каждой области двусвязности, а определение для каждого коммутатора резервного контроллера осуществляется с помощью метода Балаша.

Таким образом, во всех работах используются разные подходы к решению задачи размещения контроллеров, однако ни в одной из работ не учитывается время на синхронизацию контроллеров для нового потока в сети. Также стоит отметить, что в работах [9] и [8] постановки задач пересекаются с постановкой задачи данной статьи, однако в работе [8] авторы не публикуют алгоритм решения задачи, а методы, используемые в статье [9] неприменимы к постановке задачи данной статьи. Таким образом, задача размещения контроллеров в данной постановке задачи ранее не рассматривалась.

3. Алгоритм решения поставленной задачи

В рамках данной работы был разработан жадный алгоритм решения задачи размещения контроллеров распределённой платформы управления в ПКС сети масштаба WAN с учётом требования отказоустойчивости, в котором можно выделить 3 этапа: генерация начального размещения контроллеров, рекурсивный поиск решения, выбор наилучшего решения.

3.1 Генерация начального размещения контроллеров

Центром из k узлов сети будем называть k – коммутаторов, среднее расстояние от которых до всех коммутаторов не больше, чем у других коммутаторов. Иначе говоря, $s_1..s_k \in V$ – центр из k узлов сети, если: $\forall c \in s_1..s_k,\ \forall s \in V \setminus \{s_1..s_k\}\ \ \frac{\sum_{i=1}^{|V|} lat(c,i)}{|V|} \le \frac{\sum_{i=1}^{|V|} lat(s,i)}{|V|}$.

Под ***размещением k контроллеров в центре сети*** будем понимать размещение k контроллеров в узлах, принадлежащих центру из k узлов сети.

Так как одним из ограничений на решение является задержка в худшем случае при установлении нового потока, и одной из минимизируемых метрик является метрика средней задержки при установлении нового потока, то начальное размещение контроллеров производится в центре сети, что позволяет минимизировать время

синхронизации контроллеров, а также задержку от контроллеров до коммутаторов.

Таким образом, на первом этапе разработанного алгоритма происходит генерация начального размещения контроллеров в центре сети и занесение данного размещения в буфер рассмотренных размещений. Данный буфер нужен для того, чтобы не рассматривать одно и то же размещение несколько раз и избежать циклов в работе алгоритма.

3.2 Рекурсивный поиск решения

Два размещения контроллеров называются *соседними*, если одно из них получается из другого, перестановкой не более, чем одного контроллера не более чем на один хоп от его текущего местоположения. Размещение контроллеров x называется *предком* размещения контроллеров y, если в ходе работы рекурсивного алгоритма поиска решения на размещении x был вызван рекурсивный алгоритм на соседнем с размещением x размещении y.

После того, как получено начальное размещение контроллеров, используется рекурсивный алгоритм поиска решения. Общая идея данного алгоритма заключается в направленном рекурсивном поиске решения. Для каждого размещения проводится сравнение данного решения с предком по специальной метрике, и если данное решение не хуже предка, то происходит рекурсивная обработка всех соседних решений.

На вход алгоритму рекурсивного поиска подаётся текущее рассматриваемое размещение и размещение предок. На выходе алгоритма получаем наилучшее решение из данного направления решений. Алгоритм представлен в листинге 1, где WCLForCur – переменная, хранящая задержку в худшем случае для всех сценариев отказа, Cost – переменная, содержащая стоимость обеспечения управления сетью, AVGLat – средняя задержка на установление нового потока в сети, WCL содержит задержку в худшем случае для текущего сценария отказа, a seenPlacements – буфер рассмотренных ранее размещений контроллеров.

```
Solve(cur_sol, parent_sol) {
   WCLForCur=0;
   Cost=0;
   AVGLat=0;
   for each G_i^j ∈ G_0^0 ⋃ G_1^0 ⋃ G_0^1 ⋃ G_1^1 {
     SwitchGreedyDistribution();
      while(1){
          if ( ∃c ∈ C : load(k) < Σ_{<s,c>∈CD} load(s) ) {
              if (!LocalRedistribution())
                 exit(-1);    // нет решений
          } else break;
      }

      WCL=computeWCL();
      if ( WCL > WCLForCur )   WCLForCur = WCL;
      if ( G_i^j = G_0^0 ) {
         Cost = computeCost();
         AVGLat = computeAVGLat();
      }
}
if (isParentBetter()) return;
else {
   for each CP_i ∈ neighbor(CP_cur) and CP_i ∉ seenPlacement {
      seenPlacement ← CP_i;
      tmp = curSolution;
      curSolution = CP_i;
      Solve(tmp);
      curSolution = tmp;
   }
}
}
```

Рисунок 1. *Рекурсивный алгоритм поиска решения*

Функция switchGreedyDistribution (листинг 2) реализует жадный алгоритм распределения управления коммутаторами в сети. Функция localRedistribution (листинг 3) реализует жадный алгоритм перераспределения коммутаторов по контроллерам и возвращает true, если найдено более подходящее распределение управления. Функции computeWCL, computeCost, computeAVGLat вычисляют задержку в худшем случае на установление нового потока, стоимость обеспечения управления сетью и среднюю задержку на установление нового потока в сети соответственно для текущего сценария отказа. Функция isParentBetter (листинг 4) сравнивает текущее размещение с размещением предком и возвращает true, если размещение-предок лучше, чем текущее размещение,

функция neighbor возвращает множество размещений, являющихся соседними для текущего размещения.

После генерации начального размещения контроллеров запускается рекурсивный алгоритм поиска решения, для которого текущим решением является начальное размещение контроллеров в сети, а размещение предок отсутствует. В результате работы данного алгоритма получаем набор решений, найденных на различных направлениях рекурсивного обхода.

SwitchGreedyDistribution(){

$CD_{cur} = \varnothing;$

for each $s_i \in V$ {

$CD_{cur} \longleftarrow < s_i, c_i >: \forall c \in C, c_j \neq c_i, lat(c_i, s) \leq lat(c_j, s);$

}

}

Рисунок 2. *Алгоритм распределения управления*

LocalRedistribution() {

minLat = ∞;

min *Switch* = −1;

newCon = −1;

for each $s_i \in V :< k_i, s_i >\in CD$ {

bestCon = −1;

localMin = ∞;

newCon = k;

for each $k_j \in C, k_j \neq k_i, load(k_j) < load(s_i) + \sum_{<k_i, s> \in CD} load(s)$

{

if (localMin > lat(k_j, s_i)) {

localMin = lat(k_j, s_i);

bestCon = k_j;

}

}

if (localMin < minLat) {

min *Lat* = *localMin*;

min *Switch* = s_i;

newCon = k_j;

}

}

if (minLat == ∞) return false;

else return true;

}

Рисунок 3. *Алгоритм локального перераспределения управления*

```
isParentBetter() {
   if (parent == NULL) return false;
   if (curWCL > Lmax) {
      if (curWCL > parentWCL) return true;
      else return false;
   } else {
      if (parentWCL > Lmax or curCost < parentCost)
         return false;
      if (curCost == parentCost and curAVG <= parentAVG)
         return false;
      return true;
   }
}
```

Рисунок 4. *Алгоритм анализа текущего размещения*

3.3 Выбор наилучшего решения

В результате работы рекурсивного алгоритма может быть найдено несколько решений. Сначала отсекаются все решения, у которых задержка на установление нового потока в худшем случае превышает L_{max}. После отсечения остаются только корректные решения, и в качестве итогового решения поставленной задачи выбирается решение с минимальной стоимостью обеспечения управления сетью (метрика стоимости). Если таких решений будет несколько, то берётся то из них, у которого средняя задержка на установление нового потока наименьшая. Если и таких окажется несколько, то в качестве ответа можно брать любое решение. Если же после завершения работы алгоритма рекурсивного поиска не было получено ни одного решения, то жадный алгоритм выдаёт сообщение «не найдено решений для данных параметров» и завершается.

Таким образом, в рамках данной работы разработан жадный рекурсивный алгоритм решения поставленной задачи. На вход алгоритма подаётся топология, параметры элементов сети, а также предельная задержка на установление нового потока. На выходе данного алгоритма получаем размещение контроллеров в сети, а также распределение управления коммутаторами между контроллерами.

Данный алгоритм является завершаемым, так как количество размещений контроллеров ограничена, а из-за наличия буфера рассмотренных размещений алгоритм не будет повторно просматривать одни и те же размещения. Сложность данного алгоритма в худшем случае оценивается сложностью алгоритма полного перебора и зависит экспоненциально от количества узлов в сети. Однако, учитывая такие свойства реальных глобальных сетей, как существенные задержки на каналах связи, географическое местоположение реальных узлов сети, а также особенности постановки задачи, можно предположить, что реальная сложность разработанного алгоритма будет намного меньше. Также

стоит заметить, что найденное алгоритмом решение является корректным, так как все некорректные решения отсекаются в алгоритмах локального перераспределения управления, рекурсивного поиска решения и на этапе выбора наилучшего решения.

4. Программное средство

В рамках данной работы было разработано программное средство решающее поставленную задачу отказоустойчивого размещения контроллеров в глобальных ПКС сетях. Программное средство включает в себя реализации алгоритма полного перебора и разработанного жадного алгоритма решения задачи. Программное средство имеет графический интерфейс и способно визуализировать полученное решение с помощью программы визуализации графов Graphviz. Программное средство написано на языке программирования C++ с использованием библиотек Qt.

Для запуска программного средства необходимы библиотеки Qt-5.6. Для возможности визуализации решений требуется установленная программа Graphviz. На вход программе подаются топологии в формате Graphml, а также указанные пользователем параметры запуска программы. В ходе работы программы создаётся файл в формате CSV, в который заносятся результаты работы алгоритма на входных топологиях.

5. Экспериментальное исследование

Экспериментальное исследование проводилось на топологиях из библиотеки реальных топологий Topology Zoo [11] по двум основным направлениям: сравнение и анализ результатов работ разработанного алгоритма и алгоритма полного перебора и анализ зависимости результата работы разработанного алгоритма от входных параметров.

Все параметры запуска делились на две категории: избыток производительности и недостаток производительности и исследования проводятся отдельно на этих двух категориях входных данных. В категорию избытка производительности попадают параметры запуска с большими значения параметра L_{max}, производительности контроллеров, а также маленькие значения времени синхронизации контроллеров. В категорию недостатка производительности попадают параметры с небольшим значением L_{max} и производительности контроллера и большим временем синхронизации контроллеров. Смысл данного разделения заключается в следующем: предполагается, что при избытке производительности разработанный алгоритм должен обрабатывать большее количество размещений и иметь большую точность, а при недостатке про-

изводительности – решение будет находиться быстрее, но с меньшей точностью.

В ходе экспериментального исследования было получено, что за два часа работы разработанный алгоритм нашёл решения на топологиях с количеством узлов, не превышающих 24 и 28 для избытка производительности и недостатка производительности соответственно. Алгоритм полного перебора находит решение для топологий с количеством узлов не превышающим 11 на обеих категориях запуска за то же время. Таким образом, можно сделать вывод, что в данной постановке задачи алгоритм полного перебора следует использовать на топологиях, в которых количество узлов не превышает 11. Если же количество узлов в сети больше 11, то следует использовать разработанный алгоритм. Сравнение работ алгоритмов проводилось на топологиях с количеством узлов не превышающем 11.

Разработанный алгоритм на 18 различных параметрах запуска из категории избыточной производительности в 76% случаев нашёл решение для того же количества контроллеров, что и переборный алгоритм, а на 18 параметрах из категории недостатка производительности – в 26% случаев.

Также было получено, что отношение значений метрик средней задержки установления нового потока и стоимости разработанного алгоритма к соответствующим значениям метрик алгоритма полного перебора для избыточной производительности находятся в диапазоне от 95% до 122%, а для недостатка производительности – в диапазоне от 82% до 154%.

Исследования разработанного алгоритма на топологиях размером до 21 узла на 36 различных параметрах запуска показало, что среднее время работы алгоритма зависит линейно от количества узлов в сети.

Заключение

В рамках данной работы был разработан жадный алгоритм размещения контроллеров распределённой платформы управления в ПКС с учётом требований отказоустойчивости, а также программное средство реализующее данный алгоритм. Проведено экспериментальное исследование разработанного алгоритма на топологиях реальных сетей.

Возможны следующие направления дальнейшего исследования:

- Исследование применимости эвристических алгоритмов для решения данной задачи.
- Проведение экспериментального исследования на других входных данных и наборах реальных сетей.

Литература

1. Смелянский Р.Л. Программно-конфигурируемые сети // Открытые системы. СУБД, №09, 2012, с. 15-26.

2. Open Networking Foundation. OpenFlow Switch Specification, Version 1.3.0 (wire protocol 0x04), 2012.

3. Пашков В.Н. О подходе к обеспечению отказоустойчивости контроллера в программно-конфигурируемых сетях // Сборник тезисов XX Международной научной конференции студентов, аспирантов и молодых ученых Ломоносов-2013, секция Вычислительная математика и кибернетика, Москва, 2013, с. 35-36.

4. Пашков В.Н. Разработка высокодоступной платформы управления для программно-конфигурируемых сетей // Материалы 19-ой международной конференции по вычислительной механике и современным прикладным программным системам, ВМСППС'2015. Изд-во МАИ, Москва, 2015, с. 169–171.

5. Vasily Pashkov, Alexander Shalimov, and Ruslan Smeliansky. Controller Failover for SDN Enterprise Networks // In Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), 2014 First International, IEEE, 2014, P. 1–6.

6. Heller B., Sherwood R., McKeown N. The Controller Placement Problem // HotSDN '12 Proceedings of the first workshop on Hot topics in software defined networks, 2012, P. 7-12.

7. Francisco J. Ros, Pedro M. Ruiz. On Reliable Controller Placements in Software-Defined Networks // Computer Communications, 2016, P.41-51.

8. Francisco J. Ros, Pedro M. Ruiz Five Nines of Southbound Reliability in Software-Defined Networks // HotSDN '14 Proceedings of the third workshop on Hot topics in software defined networking, 2014, P. 31-36.

9. Hock D., Hartmann M., Gebert S., Jarschel M., Zinner T., Tran-Gia P. Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks // Teletraffic Congress (ITC), 2013.

10. Зимарина Д. Разработка и исследование метода размещения контроллеров в программно-конфигурируемых сетях с учетом требований отказоустойчивости.Дипломная работа. Лаборатория вычислительных комплексом ВМК МГУ имени М.В. Ломоносова, 2013. 67 с.

11. Internet Topology Zoo [HTML] (http://www.topology-zoo.org)

Смелянский Р.Л., Слюсарь Д.Р.

Обзор протоколов надежной многоадресной рассылки для применения в задаче управления беспроводными локальными Wi-Fi сетями

Введение

Задача организации централизованного управления сетей находит применение повсеместно – от небольших беспроводных сетей локального масштаба до магистральных проводных соединений. Контрольный трафик в сетях любого масштаба должен передаваться максимально надежным образом. И в проводных сетях с большой пропускной способностью можно добиться надежной доставки сообщений с помощью применения транспортных протоколов с подтверждением о приеме сообщений. Однако в беспроводных сетях, основанных например на Wi-Fi, происходит достаточно большое число потерь пакетов, что влечет постоянную перепосылку сообщений и уменьшение общей пропускной способности каналов связи.

При рассмотрении беспроводных Wi-Fi сетей [1], в которых для передачи контрольного трафика и клиентского используются одни и те же беспроводные каналы связи (например такое возможно при беспроводном соединении точек доступа друг с другом посредством работы в режиме ad hoc [2]), использование традиционных транспортных протоколов на основе TCP может повлечь значительную деградацию в суммарной пропускной способности беспроводной сети.

В соответствии с этим возникает задача поиска оптимального для применения в качестве транспорта контрольных сообщений в локальной беспроводной Wi-Fi сети протокола надежной рассылки с поддержкой многоадресной передачи. В данной статье проведен обзор статей для решения вышепоставленной задачи, а также описано проведенное экспериментальное исследование отобранных в обзоре протоколов передачи.

1. Обзор статей

В рамках проведения обзора был рассмотрен ряд статей [3,5,6], в которых поднимаются вопросы организации надежной многоадресной рассылки в том числе и в беспроводных сетях.

1.1 Критерии обзора

Поиск оптимального протокола производился исходя из следующих критериев:

- Протокол стандартизирован и формально описан;
- Протокол является надежным, передача пакетов гарантирована;
- Накладные расходы, измеряемые в передаваемых байтах данных, необходимых на поддержку надежности передачи малы;
- Наличие полной протестированной открытой реализации.

1.2 Протоколы транспортного уровня

Транспортный уровень является третьим уровнем в сетевом стеке TCP/IP [4]. На данном уровне осуществляется взаимодействие приложений поверх сетевого уровня, а именно с помощью IP адресов с указанием порта приложения.

Рассмотрение протоколов низших уровней не имеет смысла, вследствие того, что в Wi-Fi пока еще не поддерживается надежная многоадресная рассылка на канальном уровне.

В настоящее время существует достаточно большое количество протоколов данного уровня, одной из основных функций которых является надежная рассылка данных с поддержкой мультивещания. Среди них:

- Протокол мультивещания на основе локальных групп (LGMP, Local Group-Based Multicast Protocol);
- Протокол надежного мультивещания на основе негативных подтверждений (NORM, Nack-Oriented Reliable Multicast);
- Прагматичный протокол многоадресной рассылки(PGM, Pragmatic Multicast Pro-tocol);
- Протокол передачи файлов на основе UDP(UFTP,UDPBased FileTransfer Protocol).

Все являются надстройками над двумя базовыми протоколами транспортного уровня: UDP и TCP. Протокол TCP, предоставляя end-to-end надежную передачу сообщений, практически не используется для многоадресной рассылки из-за больших накладных расходов на обеспечение надежности.

LGMP (Local Group-Based Multicast Protocol [5]) – протокол мультивещания, поддерживающий как полную надежность передачи, так и надежность передачи данных с некоторой вероятностью.

В своей основе протокол LGMP использует разбиение множества получателей на локальные подгруппы, в рамках которых происходит восстановление и пересылка потерянных пакетов. В группах выбирается узел, называемый локальным контроллером. Локальный контроллер взаимодействует с источником данных и запрашивает перепосылку данных в случае утери. Базовым протоколом является UDP и сообщения о подтверждении (ACK), которые передаются от локальных

контроллеров источнику и от локальных узлов до своего локального контроллера.

LGMP хорошо подходит для глобальных сетей. В локальных сетях, а тем более в локальных беспроводных сетях, использование ACK пакетов влечет большую нагрузку на каналы передачи. Также это влечет слабую масштабируемость (в смысле малого числа локальных групп и малого числа узлов в каждой из групп) при отсутствии возможности нарастить производительность источника данных и локальных контроллеров.

Также очевидные минусы LGMP: отсутствие стабильной реализации в совокупности с отсутствием истории использования в коммерческих проектах.

NORM (Nack-Oriented Reliable Multicast [5]) – протокол надежной передачи трафика прикладного уровня, финальная версия зафиксирована в RFC 5740 в 2009 году.

Протокол NORM разработан для надежной передачи трафика с легкой масштабируемостью числа приемников. Основан на передаче специальных NACK (Negative Acknowledgement) пакетов от получателей отправителю в случае неполучения пакетов. Тем самым горизонтальная масштабируемость сети получателей в условиях надежных каналов передачи в сети практически ничем не ограничена. Однако в протоколе заложена возможность опроса некоторых узлов для получения пакетов о подтверждении принятия трафика (ACK).

NORM работает поверх UDP сокетов. Согласно RFC, благодаря грамотно настроенным интервалам и включенной поддержкой управления перегрузкой, удается достигнуть хороших скоростей при многоадресной передачи контента. Однако на практике (См. [?]) протокол NORM показывает себя не лучшим образом, значительно теряя пропускную способность при небольших потерях в канале.

На текущий момент существуют две независимые реализации протокола: от Naval Research Laboratory (NLR) [8] и от French National Institute for Research in Computer Science and Control (MCL). Первый (NORM/-NLR) является более развитым, библиотека написана на C++ и имеет хорошую документацию.

PGM (Pragmatic Multicast Protocol [5]) описан в RFC 3208. Основным разработчиком является компания Cisco Systems Inc. Основной идеей протокола является надежная доставка контента максимально возможному числу клиентов. Для этого сразу было принято решение в пользу использование Negative-ACK, как и в протоколе NORM.

При инициализации работы PGM, пользователю предлагается выбрать начальный размер окна, который будет использован при передаче. По-умолчанию это 10 Мбайт. Есть возможность динамического расширения размеров окна на основе стабильности

передачи (если давно не было NACK сообщений, то окно увеличивается). PGM работает поверх UDP.

Существует открытая реализация протокола – openPGM [7], написанная на C++.

UFTP (UDP-Based File Transfer Protocol with Multicast) – протокол многоадресной передачи данных, ориентированный на надежную передачу больших файлов.

Перед передачей на источнике данных происходит дробление исходного файла на разделы. Один раздел представляет из себя несколько пакетов (один пакет UFTP гарантированно вмещается в один UDP пакет). Далее передача осуществляется по разделам – после передачи которых происходит синхронизация. Если потери есть, то серией одноадресных передач источник передает потерянные пакеты. Итого, если раздел с номером N полностью получен всеми адресатами, то происходит передача раздела с номером N+1.

Протокол UFTP хорошо подходит для передачи больших данных. Исходя из экспериментальных исследований [?], UFTP в среднем много лучше по пропускной способности, нежели PGM или NORM. Однако когда объем трафика невелик, а также существует ограничения, например, на энергопотребление, накладные расходы нивелируют выигрыш в пропускной способности.

В сети существует достаточное число стабильных реализаций, в том числе на C++ и Python.

1.3 Протоколы прикладного уровня

Протоколы прикладного уровня, несмотря на их большие задержки и большее время реагирования в случае потери пакета, повсеместно используются для организации транспорта. Дело в том, что основным преимуществом протоколов прикладного уровня над низкоуровневыми протоколами является скорость разработки сетевого программного обеспечения.

Между тем некоторые протоколы создавались как альтернатива протоколам транспортного уровня. К таким протоколам, например, относятся CoAP и MQTT, созданные для работы с маломощными устройствами из Интернета вещей (IoT, Internet of Things).

CoAP (Constrained Application Protocol [6]) – ограниченный прикладной протокол передачи данных – синхронный протокол запросов/ответов, разработанный в Internet Engineering Task Force (IETF) для управления устройствами с ограниченными ресурсами (сенсорами, датчиками и пр.).

CoAP похож на HTTP, только является его облегченной версией, нетребователен к ресурсам. Работает поверх UDP. Есть четыре вида команд: GET, PUT, POST и DELETE. По-умолчанию в CoAP отсутствует

надежность, однако можно достаточно легко включить механизм обеспечения надежности с помощью двух битов для QoS.

CoAP поддерживает многоадресную рассылку, однако есть некоторые ограничения, связанные с безопасностью. Дело в том, что по-умолчанию CoAP не шифрует трафик. Сделано это чтобы сократить энергозатраты и затраты на передачу данных. Однако можно поверх CoAP использовать DTLS (Datagram Transport Layer Security), что исключает поддержку многоадресной рассылки и влечет дополнительные траты на установку сессий между клиентами и серверами, что повышает энергозатраты.

В настоящее время существует много открытых реализаций, написанных на различных языках, протокола CoAP для абсолютно разных устройств и разных архитектур (например [9]).

MQTT (Message Queue Telemetry Transport [6])– прикладной протокол передачи данных, разработанный в IBM специально для применения в Интернете Вещей (сенсоров и датчиков, IoT). Несмотря на то, что MQTT использует TCP в качестве базового протокола передачи данных, энергоэффективность и практичность заложены в архитектуру. Для этого, поверх TCP MQTT включает систему публикаций (publish) и подписок (subscribe).

MQTT представляет из себя платформу, состоящую из брокера сообщений (сервера) и клиентов. Брокер содержит некоторый набор тем. Все клиенты, участвующие в передаче данных, регистрируются на брокере и могут публиковать данные в определенную тему или темы, либо подписываться на получение данных. Брокер фиксирует, когда публикуются данные и рассылает данные всем подписчикам. При этом существует несколько уровней QoS:

- Fire and forget – сообщение отправляется единожды без подтверждения о доставке.
- Delivered at list once – сообщение гарантировано будет доставлено всем подписчикам, но не гарантируется, что единожды.
- Delivered exactly once – сообщение гарантировано будет доставлено единожды. Для достижение этого используется четырехстороннее рукопожатие.

Применение схемы публикаций и подписок больше подходит для применения в IoT, нежели схема запрос-ответ, используемая в CoAP, из-за уменьшенной нагрузки на канал передачи данных, а также уменьшенной обработки сообщений.

С точки зрения безопасности, MQTT более защищен, так как применяется стандартное TLS/SSL шифрование.

MQTT имеет большие накладные расходы, по сравнению с CoAP из-за использования TCP в качестве базового транспортного протокола. Однако показано [6], что при небольших потерях в канале MQTT имеет

меньшие задержки, по сравнению с CoAP. Однако при увеличении уровня QoS оба протокола значительно увеличивают трафик, уменьшая тем самым пропускную способность канала передачи.

Уровень модели	Имя протокола	Наличие стандарта (описания)	Обеспечение надежной передачи	Накладные служебные расходы	Наличие открытой полной реализации
Транспортный	LGMP	-	+	Большие	Нет
	NORM	+	+	Низкие	Есть
	PGM	+	+	Низкие	Есть
	UFTP	-	+	Средние	Есть
Прикладной	CoAP	+	+	Средние	Есть
	MQTT	+	+ (QoS 1)	Большие	Есть

Таблица 1. *Сводная таблица по протоколам надежной многоадресной рассылки*

1.4 Результаты обзора и выводы

В соответствии с выбранными критериями обзора, была сформирована таблица 1 по описанным протоколам, призванным обеспечить надежный транспорт с поддержкой многоадресной рассылки.

Согласно таблице, наиболее подходящими для решения задачи организации надежной многоадресной рассылки в условиях локальной беспроводной Wi-Fi сети можно считать протоколы PGM, NORM, MQTT и CoAP и выбрать оптимальный исходя из результатов экспериментального тестирования.

2 Экспериментальное исследование

2.1 Описание тестового стенда

Для проведения тестирования выделенных в обзоре протоколов был собран реальный стенд, эмулирующий работу Wi-Fi сети. В стенде используются четыре виртуальные машины на базе OS Ubuntu, к каждой из которых подключен радио интерфейс Wi-Fi (TP-Link TL-WN7200ND) и установлено соответсвующее ПО для его управления.

На одной виртуальной машине развернута программа управления беспроводными сетями в контрольном режиме. На

остальных ПО запущено в режиме клиента. Соответственно виртуальные машины объединены в единую беспроводную сеть (режим Wi-Fi ad hoc, на схеме mesh). Схема представлена на Рис. 1.

При проведении тестирования на каждой виртуальной машине собирается трафик, проходящий через сетевой интерфейс wifi, а также замеряется суммарное время работы ПО при обновлении конфигурации Wi-Fi сети.

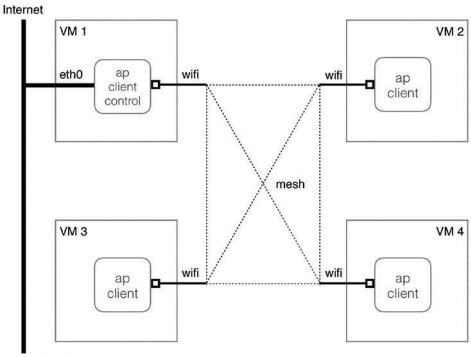

Рисунок 1. *Схема экспериментального стенда*

2.2 Критерии и методика тестирования

В ходе тестирования необходимо:

- Измерить суммарное количество трафика, прошедшего через сеть при пересылке контрольных сообщений об обновлении конфигурации сети;
- Эмулировать отказ соединений (сброс части пакетов);
- Измерить время на обновление сети;
- Сравнить работу протоколов PGM, NORM, MQTT и CoAP (реализация протоколов взята из [7, 8, 9, 10]).

В результате для сбора экспериментальных данных была выбрана методика тестирования с использованием tcpdump и файла статистики по сетевым интерфейсам (/proc/net/dev) и проведена серия тестов (5 запусков каждого протокола при каждом значении надежности сети).

Рисунок 2. *График зависимости времени обновления сети от надежности сети и протокола передачи*

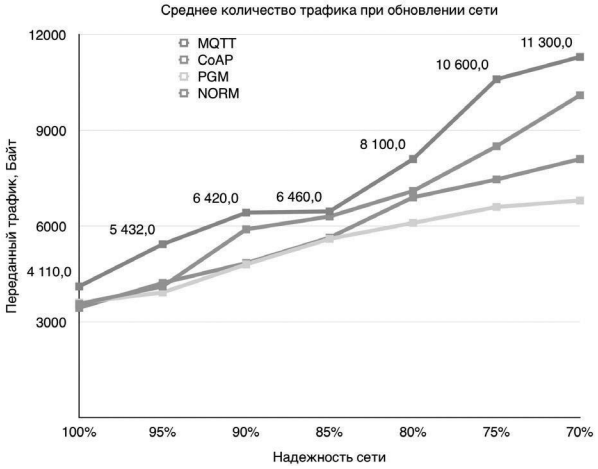

Рисунок 3. *График зависимости переданного числа трафика в байтах от надежности сети*

2.3 Результаты тестирования

На риснуках 2 и 3 представлены усредненные результаты экспериментальных прогонов на тестовом стенде. Как видно из графиков, оптимальным протоколом как с точки зрения затраченного времени на обновление сети, так и с точки зрения накладных расходов, является протокол PGM.

3. Заключение

Задача централизованного управления беспроводными Wi-Fi сетями в настоящее время является достаточно востребованной. Вследствие ограниченной пропускной способности и недостаточной канальной надежности, контрольный трафик требует использования надежных транспортных протоколов.

В данной статье показано, что транспортный протокол PGM является оптимальным для доставки управляющего трафика в локальных беспроводных Wi-Fi сетях исходя из накладных расходов.

Литература

1. IEEE Std 802.11 Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. 2012 [PDF] URL: (http://standards.ieee.org/getieee802/download/802.112012.pdf)
2. Hiertz G. R. et al. Principles of IEEE 802.11s //Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on. – IEEE, 2007. – С. 1002-1007.
3. Цыганова А. Надежная многоадресная рассылка в беспроводной меш-сети //Труды конференции "Информационные технологии и системы"(ИТИС-2009). – 2009.
4. Смелянский Р. Л. Компьютерные сети: в 2 т. Т. 2: Сети ЭВМ: учебник для студентов вузов, обучающихся по направлениям 010400"Прикладная математика и информатика"и 010300"Фундаментальная информатика и информационные технологии"//М.: Академия. – 2011.
5. Siemens E., Bakharev A. Evaluation of reliable multicast implementations with proposed adaptation for delivery of big data in wide area networks //Proceedings of ICNS 2013, the Ninth International Conference on Networking and Services. – 2013. – P. 160-164.
6. Karagiannis V. et al. A survey on application layer protocols for the internet of things //Transaction on IoT and Cloud Computing. – 2015. – В. 3. – N. 1. – P. 11-17.
7. Steven McCoy Open realisation of PGM algorithm, 20032017. URL: (https://github.com/steve-o/openpgm)
8. US Naval Research Laboratory Nack-Oriented Reliable Multicast protocol realisation, 2004. URL: (https://goo.gl/MrvGEg)
9. C-Implementation of CoAP, URL: (https://libcoap.net/)
10. MQTT C Client, Eclipse Open Sourse, Community, URL: (https://eclipse.org/paho/clients/c/)

Швецов Д.А., Шалимов А.В.

Система автоматической генерации правил в программно-конфигурируемых сетях

Введение

Программно-конфигурируемые сети обладают большим преимуществом по сравнению с традиционными — видение всей сети, новые сервисы, простота управления. Но программирование, с использованием низкоуровневых протоколов управления, это сложная задача, которая требует заботы о многих низкоуровневых деталях, не позволяет сосредоточиться на решении поставленной проблемы и делающая практически невозможной независимость приложений друг от друга. Такую разработку можно сравнить с программированием на языке ассемблера. И также как появились высокоуровневые языки программирования общего назначения такие, как C/C++, Java, Python, появляются языки программирования для приложений ПКС (Procera, Piretic, Maple [1,2,3]). Однако эти языки программирования имеют недостатки – низкая выразительность, перегруженный синтаксис или большие накладные расходы, выраженные в большом числе сообщений управления, плюс эти языки нацелены на программирование отдельно взятого сетевого устройства, забывая о преимуществах ПКС — видение всей сети, в результате программы становятся менее эффективными, а программный код объемнее и менее понятным. Поэтому разработка специализированных языков программирования, упрощающих разработку сетевых приложений для таких сетей, остается открытой задачей.

В данной работе представлено усовершенствование системы Maple[3] для OpenFlow[4] сетей для решения задачи повышения эффективности и повышения абстракции языков программирования для приложений ПКС. Данная система была выбрана потому что она предоставляет удобную абстракцию для программирования в императивной парадигме — программист задает алгоритм обработки появившихся в сети пакетов.

1. Система Maple

Maple предлагает абстракцию «видеть все пакеты в сети», которая заключается в том, что программист описывает на языке общего назначения функцию-обработчик, которая на вход получает все пакеты, который появились в сети, а результатом является действие,

которое необходимо выполнить над этими пакетома. Такая абстракция проще, чем абстракция, предлагаемая OpenFlow, когда надо запоминать состояние правил на коммутаторах и состояние приложений на контроллере. Таким образом проргаммист избавляется от необходимости манипулировать деталями низкого уровня, такими как таблицы потоков, приоритеты и поля соответствия.

Самая простая реализация данной абстракция заключается в том, что надо отправлять *каждый* пакет в сети на контроллер, вызывать функцию-обработчик и выполнять полученные действия над этим пакетом. Но накладные расходы на использование такого метода превышают преимущества использования такой абстракции – отправлять все пакеты сети на контроллер через каналы управления недопустимо.

Тогда для повышения производительности Maple использует кэширование выполненных действий функций-обработчиков – на практике для большинства пакетов, например, с одинаковыми заголовками выполнение функции-обработчика идет по одинаковым путям. Например, приложение может отправлять пакеты по одному и тому же маршруту, если у них одинаковые MAC адреса назначения. Следовательно, мы вызовем функцию для первого пакета, а когда придет второй пакет с такими же MAC адресами, результат первого выполнения функции можно повторно использовать для второго пакета. Основная цель оптимизации кэширования — это повторное использование действий из прошлых выполнений функции.

Maple предоставляет следующий интерфейс для взаимодействия с пакетом:

1. прочитать поле пакета: метод load;
2. проверить поле пакета на соответствие заданному константному значению: метод test;
3. выбрать действие, которое необходимо произвести над пакетом: переслать пакет на заданный порт или порты, отбросить пакет, отправить на контроллер.

Метод используемый для кэширования потенциально сложной функции f заключается в том, чтобы записать те действия, которые повлияли на результат ее выполнения: обращение к входным данным (load и test методы). Последовательность таких обращений называется трассой. Так же в эту трассу записывается результат функции.

Ключевая структура данных, которой управляет Maple, для того чтобы представить собранные трассы, это *Trace Tree*. Trace Tree обеспечивает абстрактное, частичное представление политики, выраженной функцией-обработчиком. Будем считать, что пакет имеет поля $a_1, \ldots a_n$, и записывать как $p.a$ значения поля a пакета p. Будем обозначать через $dom(a)$ набор возможных значений поля a: $p.a \in dom(a)$ для любого пакета p и любого поля a.

Определение. TraceTree — это корневое дерево, в котором каждой вершине t соответствует поле $type_t$, чье значение может быть одним из следующих: *L(Leaf), V(Value), T(test)*

1. Если $type_t = L$, то вершина t содержит поле $value_t$, которое представляет некоторое возможное возвращаемое значение функцией-обработчиком. Эта вершина представляет поведение программы, которая возвращает $value_t$ без дальнейшего исследования пакета.
2. Если $type_t = V$, то t содержит поля $attr_t$ и $subtree_t$, где $subtree_t$ — это ассоциативный массив, такой что $subtree_t[v]$ указывает на поддерево *TraceTree*, а $v \in keys(subtree_t)$. Эта вершина представляет поведение программы, в которой если пакет удовлетворяет условию $p.attr_t = v$, то она продолжается в $subtree_t[v]$.
3. Если $type_t = T$, то t содержит поля $attr_t$, $value_t$, такие что $value_t \in dom(attr_t)$, и два поддерева t_+ и t_-. Этот узел отражает поведение программы, которая проверяет условие $p.attr_t = value_t$ для пакета p и идет в ветвь t_+, если условие выполняется, или в ветвь t_-, иначе.
4. Если $type\_t = \Omega$, то t не имеет полей. Этот узел представляет произвольное поведение (т.е. неизвестный результат).

Для каждого коммутатора существует свой экземпляр дерева TraceTree.

По дереву TraceTree можно произвести поиск действия для данного пакета или обнаружить, что дерево не содержит в себе информацию о решении функции для пакета.

Для каждого приложения построение начинается с пустого дерева представленного, как Ω. После сбора трассы Maple представляет ее как ветвь и добавляет к дереву TraceTree. Алгоритм проходит по дереву TraceTree и трассе одновременно, чтобы найти место для добавлении ветви. К найденному месту он добавляет оставшуюся частью
трассы.

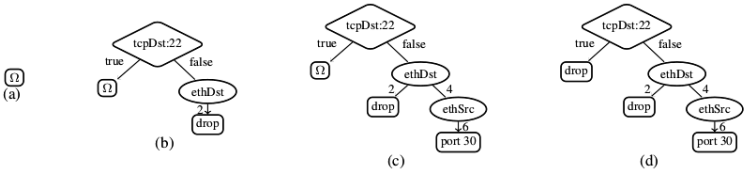

Рисунок 1: *Пример создание дерева*

Кэширование функции-обработчика в TraceTree избавляет от издержек, вызванных исполнением функций-обработчиков, но, если контроллер по-прежнему обрабатывает каждый пакет централизованно, система остается не масштабируемой. Для достижения большей

эффективности контроллеру необходимо уметь «загружать» решения приложений на OpenFlow коммутатор для быстрой обработки пакетов «на месте». Для достижения этой цели Maple компилирует TraceTree в таблицу потоков.

```
Function BuildFT(t)
   priority ← 0;
   BUILD(t, any);
   return
Function BUILD(t, m)
  if type_t = L then
     emitRule(priority, m, value_t);
     priority ← priority + 1;
  end
  else if type_t = V then
     foreach v in keys(subtrees_t) do
        BUILD(subtrees_t[v],
                    m ^ (attr_t : v));
     end
  end
  else if type_t = T then
     BUILD(t_negative, m);
     m_t ← m ^ (attr_t : value_t);
     emitRule(priority, m_t,
                    ToController)
     priority ← priority + 1
     BUILD(t_positive, m_t);
  end
```

Алгоритм 1: Построение таблицы потоков по дереву TraceTree

Match	Prio	action
ethDst:4 ethSrc:6	0	Port:30
ethDst:2	1	drop
tcpDst:22	2	toController

Таблица 1: Таблица потоков для дерева из рисунка 1 (d)

Компиляция таблицы представляет из себя рекурсивный обход дерева, на каждом узле которого изменяется множество match, представляющее из себя набор полей и значений, по которым определяется поток. В алгоритм 1 описывается функция BuildFT, реализующая компиляцию таблицы потоков.

Первым делом функция BuildFT(t) инициализирует глобальную переменную *priority* нулем и вызывает рекурсивную функцию BUILD. Функция BUILD принимает не только текущую вершину дерева TraceTree, но также множество *match*. Функция BuildFT вызывает BUILD с множеством *match* равным *any* (то есть соответствует любому пакету).

Второй важной переменной является *priority*, которая увеличивается после каждой установки правил таблицы потока.

Обработка узла t соответствует его типу. Во-первых, на узле Leaf функция BUILD устанавливает правило с полями из множества match и действием, соответствующим узлу. Во-вторых, в узле Value функция BUILD рекурсивно устанавливает правила для каждой ветви поддерева, предварительно пересекая текущее множество *match* с элементом поле-значение (результат пересечения соответствует всем пакетам, которые соответствуют множеству *match* и у которых поле $attr_t$ равняется значению $value_t$).

В-третьих, узел Test. О нем можно думать, как об узле Value, но всего с двумя ветвями. К сожалению, таблица потоков не поддерживает отрицания, и, следовательно, функция BUILD не может включить отрицательное состояние в множество *match*, когда он обходит отрицательную ветвь. Чтобы справиться с этим, используется следующая техника. Для узла t устанавливается правило, которое называется барьерным и содержит действие *ToController* и $match = m_t$, являющегося пересечением элемента поле-значение указанного в Test узле и текущем множеством *match*. Барьерное правило имеет более высокий приоритет, чем правила, установленные в отрицательной ветви. Чтобы избежать блокировки барьерным правилом правил, сгенерированных в положительной ветви, барьерное правило должно иметь приоритет ниже, чем правила из положительной ветви. Формально, r_b — барьерное правило, r_- — правило из отрицательной ветви, r_+ — правило из положительной ветви. Результат работы алгоритма должен удовлетворять следующему ограничению:

$$priority(r_-) < priority(p_b) < priority(r_+)$$

В таблице 1 предоставлен пример таблицы потоков, построенный по дереву TraceTree на рисунке 1 (d).

2. Проблема частого обновления таблиц потоков и предложенный способ решения

После добавления новой ветви в дерево, необходимо загрузить правило на коммутатор, но поскольку алгоритм не гарантирует того, что приоритеты остальных правил не изменятся, необходимо обновить таблицу потоков на коммутаторе. Протокол OpenFlow не позволяет менять приоритет в правилах, и поэтому, чтобы сохранить корректность таблицы потоков, придется удалять правила из таблицы и устанавливать заново с новыми приоритетами. Все это увеличивает нагрузку на канал между коммутатором и контроллером и на процессор коммутатора. Чтобы избавиться от этих накладных расходов, надо так модифицировать алгоритм, чтобы при добавлении ветви и

последующей компиляции таблицы потоков приоритет предыдущих правил не изменился.

Предложенный алгоритм заключается в рекурсивном обходе дерева и выделении пространства приоритетов для вершин, каждый из которых удовлетворяет вышеописанному условию для всех Test вершин выше данной вершины.

Функция SetPrio, описанная в алгоритме 2, вызывает функцию SET для корня с пространством приоритетов от нуля до максимально возможного приоритета. Функция SET ведет себя по-разному для разных типов вершин:

1. листовой вершине присваивается приоритет равный середине полученного отрезка,
2. для Value вершины функция рекурсивно вызывается для каждого дочернего узла с тем же самым пространством приоритетов так, как правила из дочерних вершин не будут пересекаться как минимум по читаемому полю.
3. Test вершине присваивается приоритет равный середине отрезка, после чего происходит рекурсивный вызов для отрицательной ветви с приоритетами меньше присвоенного, а для положительной с приоритетами больше присвоенного. Таким образом для всех правил из отрицательной ветви приоритет будет меньше присвоенного, который меньше чем приоритет правил из положительной ветви

Иллюстрация алгоритма представлена на рисунке 2

```
Function SetPrio(root)
   SET(root, 0, N)
Function SET(t, p1, p2)
   if type_t = Leaf then
      if t_p unset then t_p ← (p1 + p2) /2;
   end
   else if type_t = Value then
      foreach v in keys(subtrees_t) do
            SET(subtrees_t[v], p1, p2)
      end
   end
   else if type_t = Test then
      if t_p unset then t_p ← (p1 + p2)/2
      SET(t_negative, p1, t_p – 1)
      SET(t_positive, t_p + 1, p2)
   end
```

Алгоритм 2: Выставление приоритетов

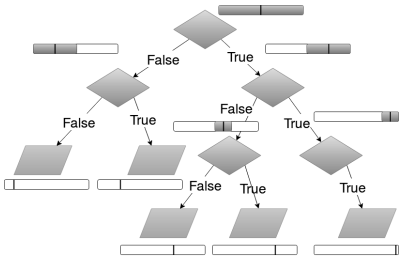

Рисунок 2: Пример приоритетов, выставленных алгоритмом

В результате выполнения алгоритма каждой вершине будет выдано значение, которое не будет меняться при добавлении ветвей.

Однако при работе вышеописанного алгоритма может сложиться такая ситуация, когда невозможно выдать вершине приоритет, не нарушая условия, хотя есть большое количество неиспользованных приоритетов. Такое может случится, если дерево «разрастется» в одну сторону.

Для исправления этого недостатка воспользуемся алгоритмом, который перераспределяет приоритеты в соответствии с некоторой функцией весов вершины. Для начала определим функцию веса вершины (weight) как количество вершин в поддереве, для которых необходимо выставление приоритета. Далее проведем перераспределение пространства приоритетов так, чтобы выдать более «тяжелым» вершинам (определенным с помощью функции веса) большее пространство приоритетов. После выполнения данного алгоритма придется полностью обновить таблицу потоков на коммутаторе, однако, во-первых, данный алгоритм придется применять в очень редких случаях. В OpenFlow приоритет — это неотрицательное число от 0 до 65536, и чтобы воссоздать ситуацию нехватки приоритетов необходимы 16 последовательных вызовов Test метода, что в реальной жизни встречается редко, во-вторых, после перераспределения приоритетов можно опять пользоваться алгоритмом 1 выставления приоритета в силу того, что в нем мы присваивали приоритет вершине, только если он не был определен до этого, следовательно, алгоритм не нарушит сбалансированность приоритетов.

3. Задание политик по управлению сетью

Maple предоставляет интерфейс для задания политики по управлению отдельно взятого коммутатора. И несмотря на то, что ПКС предоставляет возможности видения целой сети, например, при пересылке пакета контроллеру может быть известна вся топология сети и маршрут от начальной до конечной точки, Maple не позволяет проложить этот маршрут сразу. Вместо этого функция-обработчик будет вызвана на каждом коммутаторе, и каждый раз функция должна вернуть действие — переслать пакет на порт к следующему коммутатору в маршруте, что приведет к излишним PacketIn сообщениям. Проблема заключается в том, что поведение политик может быть специфичным для некоторых отдельных коммутаторов, и если не учитывать этот факт, то результат работы системы может не соответствовать политикам, которые выразили программисты. В следствии этого каждый коммутатор имеет свою функцию-обработчик и свое дерево TraceTree.

Продемонстрируем данную проблему на примере: Два программиста независимо решают задачи пересылки траффика между хостами и зеркалирование пакетов некоторого *конкретного* коммутатора *s1* на сервер. Если программист, решающий задачу пересылки траффика будет устанавливать правила на все коммутаторы в маршруте при первом PacketIn сообщений, то для приложения второго программиста может никогда не прийти PacketIn сообщение с коммутатора *s1*, и пакеты не будут отправляться на сервер.

Основания идея нашего решения данной проблемы заключается в объединении функций обработчиков и деревьев TraceTree разных коммутаторов в одно дерево, для этого выделим два основных пункта:

1. введение абстракции: маршрут через сеть;
2. введение поля пакета switch_id;

Абстракция маршрут через сеть, представленная набором коммутаторов и его входными и выходными портами, позволит задавать политики для целой сети, а поле пакета switch_id позволит определить специфичное поведение функции для заданных коммутаторов и гарантирует соблюдение политики. Таким образом, для вершины t: $type_t$ = L значение $action_t$ представляет из себя ассоциативный массив, такой что $action_t$[d] означает действие на коммутаторе d.

На рисунке 3 показан пример объединенного дерева с использованием поля switch_id. Чтобы задать специфичное поведение для какого-то коммутатора программисту достаточно прочитать поле switch_id у пакета и выполнить необходимую логику, если результат будет подходящим.

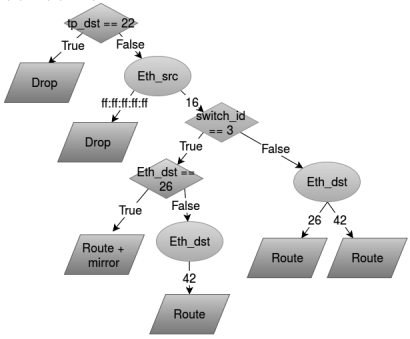

Рисунок 3: Пример объединенного дерева

Maple будет брать на себя задачу определение того, на какие коммутаторы необходимо поставить правила. Например, для данного примера если пакет пришел на коммутатор 1 и у данного пакета tcpDst:21, ethSrc:16, ethDst:26, и маршрут, полученный в результате выполнения функции, проходит через третий коммутатор то, несмотря на это, правила будут установлены на все коммутаторы в маршруте

кроме третьего. Далее, когда пакет дойдет до третьего коммутатора опять будет отправлено PacketIn сообщение, выполнение пройдет по другой ветви, и на данный коммутатор будет установлено правило с зеркалированием.

Алгоритм 3 BuildFlowTables описывает построение таблиц потоков для всех коммутаторов. От исходного алгоритма он отличается тем, что, во-первых, устанавливает потоки с приоритетом, выставленным заранее алгоритмом выставления приоритетов, во-вторых, он определяет множество коммутаторов, на которые необходимо установить правила, и устанавливает эти правила с соответствующим действием.

```
    Function BuildAllFlowTables(t)
      BUILD2(t, any, Allswitches)

Function BUILD2(t, m, S)
   if type_t = L then BUILDLeaf(t,m,S)
   else if type_t = V then BUILDValue(t, m, S)
   else if type_t = T then BUILDTest(t, m, S)


Function BUILDValue(t, m, S)
 if attr_t is switch_id then
    foreach d in keys(subtrees_t) do
      BUILD2(subtree_t[d], m, S ^ (attr_t : d))
    end
  else
    foreach v in keys(subtrees_t) do
      BUILD2(subtree_t[v], m ^ attr_t : v), S)
    end
end

Function BUILDTest(t, m, S)
 if attr_t is switch_id then
   BUILD2(t_negative, m, S \ (attr_t : value_t))
   BUILD2(t_positive, m, S ^ (attr_t : value_t))
 else
   BUILD2(t_negative, m, S)
    m_t ← m ^ (attr_t : value_t)
    foreach d in S do
      emitRule(d, t_p, m_t, ToController)
   end
  BUILD2(t_positive, m_t, S)
 end

Function BUILDLeaf(t, m, S)
  foreach d in keys(value_t) ^ S do
     emitRule(d, t_p, m_t, value_t[d]);
end
```

Алгоритм 3: Построение таблиц потоков по дереву TraceTree с switch_id полем

Данный метод позволяет задавать политики для целой сети в виде маршрутов и позволяет сохранить корректность в случае специфичного поведения некоторых коммутаторов.

У данного подхода есть проблема, связанная с чтением значения входного порта. Для приложения, которое во время своей работы применяет метод load к полю входного порта пакета, и результатом которого является маршрут через сеть, в ходе построения таблиц потоков для всех коммутаторов на маршруте в полях соответствия будет присутствовать поле входного порта со значением из первого коммутатора. При этом значение входного порта на следующих коммутаторах могут отличаться, и таблицы потоков не будут иметь смысла.

Есть два простых способа борьбы с данной проблемой:

1. не добавлять поле входного порта в поля соответствия таблиц потоков. Однако, такой способ не позволит, например, реализовать миграцию конечных устройств, так как, если оно подключиться к другому порту того же коммутатора, пакеты будут обрабатываться по существующей таблице потоков, и контроллер не узнает о смене местоположения;
2. при чтении входного порта неявно читать поле switch_id — это гарантирует установку правила только на один коммутатор. Но в таком случае теряется преимущество подхода с объединенным деревом, потому что с оставшихся коммутаторов в маршруте будет так же послано PacketIn сообщения на данный пакет.

Идея решения этой задачи исходит из того, что можно получить информацию о входных портах в других коммутаторах на маршруте из самого маршрута (так как он задается списком коммутаторов и его *входными* и выходными портами).

Для этого введем новый метод для доступа к полям пакета — vload, который будет использоваться только для доступа к полю входного порта. Данный метод позволяет прочитать входной порт пакета на коммутаторе, но будет устанавливать на остальные коммутаторы в маршруте правила с значением входного порта, исходя из решения, которое будет получено после выполнения функции. Применяя метод vload, программист соглашается с условием, что полученное значение имеет смысл только на коммутаторе, с которого пришло PacketIn сообщение и может быть изменено на пути следования пакета.

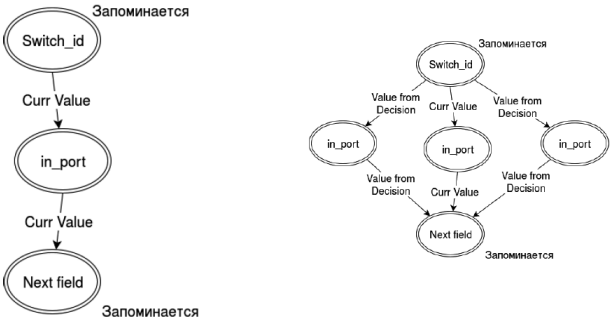

Рисунок 4: Реализация метода vload

а) при вызове метода б) после получения результата функции

Работа данного метода состоит из двух этапах:

1. при вызове данного метода сначала происходит неявный вызов метода load(switch_id), и запоминается вершина t_1, соответствующая этому вызову. После этого происходит вызов load(inPort), и результат возвращается программисту. Также необходимо будет запомнить вершину t_2, следующую за вершиной, соответствующей вершине load(inPort);
2. после выполнения функции-обработчика из ее результата берется набор коммутаторов и соответствующих входных портов DP. Далее, для каждой пары (d, p) принадлежащей DP из вершины t_1 проводим ребро со значением d к новой вершине V(inPort), из которой, в свою очередь, проводится ребро по значению p в вершину t_2. После этого полученная трасса накладывается на дерево TraceTree.

Иллюстрация реализации метода показана на рисунке 4.

К полученному направленному ациклическому графу применяется алгоритм компиляции таблиц потоков с помощью рекурсивного обхода, и таким образом получаются корректные таблицы на разных коммутаторах в сети.

4. Реализация предложенных методов

Реализация данной модификации системы Maple была произведена в контроллере RUNOS [5], который написан на языке C++ с использованием библиотек Boost, Qt и LibFluid. Данный контроллер

был выбран, потому что в нем уже была реализована система Maple и он является OpenFlow контроллером с открытым исходным кодом, распространяется под лицензией Apache License 2.0.
Особенности системы Maple в контроллере RUNOS:

1. реализована оптимизация сжатия трассы, которая заключается в исключении из трассы вершин, результат которых был уже получен при предыдущих чтениях полей пакета [3];
2. Возможность использования в методе test логического выражения с операциями «логическое и» и «логическое или», при этом система Maple "видит" выражение целиком и оптимизирует по количество правил в таблице потоков и количество выполнений функции-обработчика [6];
3. построение дерева происходит на ходу: то есть совмещены процессы создания трассы и добавления ее к дереву;
4. есть возможность читать поля пакета по маске.

Скорость работы алгоритма генерации таблицы потоков с использованием метода vload можно оценить как O(NM), где N — это количество вершин в графе, а M — количество коммутаторов в сети. Эта оценка получена из того, что для выполнения алгоритма необходимо обойти каждую вершину, при этом некоторые вершины придется обходить несколько раз, но число обходов не может превышать количество коммутаторов в сети в силу реализации метода vload. Также при реализации была произведена оптимизация по обходу дерева TraceTree — обходится не все дерево, а только построенная ветвь, что позволяет улучшить оценку до O(DM), где D — максимальная глубина дерева.

Алгоритм расстановки приоритетов можно оценить как O(N), так как для каждой вершины нужно определить приоритет, но с учетом того, что после добавлении ветви приоритет будет не выставлен только у одной вершины, а у остальных он не измениться, была достигнута сложность O(D).

Алгоритм перераспределения приоритетов можно выполнить в два шага: определение весов каждой вершины, выставление приоритетов вершинам, и может быть оценен как O(N).

5. Экспериментальное сравнение исходной системы Maple и системы с предложенными изменениями

Целью экспериментального исследования было сравнение модифицированный и исходной системы Maple по количеству системных OpenFlow сообщений (PacketIn и FlowMod) между коммутаторами в сети и контроллером. Исследование проходило в

система эмуляции компьютерной сети Mininet, использовались программные коммутаторы Open vSwitch.

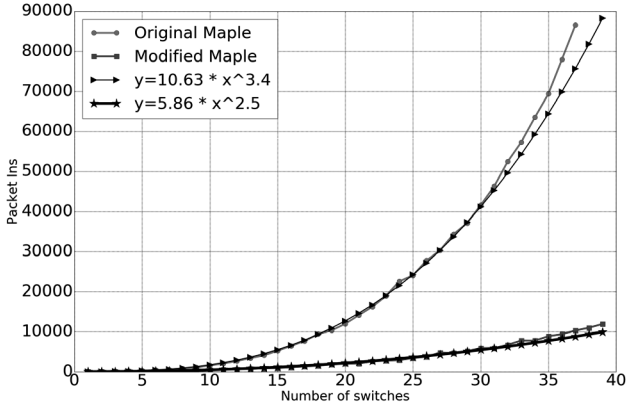

Рисунок 5: График зависимости количества PacketIn сообщений от количества конечных устподройств в сети

Тестирование происходило на разных бинарных сбалансированных топологиях с разным количеством сетевых устройств. Было проведено тестирования для количества конечных устройств от двух до сорока с максимальной глубиной топологии равной 5.

Политики по управлению сети задавались приложениями:

1. LinkDiscovery — приложение, обеспечивающие поддержку информации о соединениях в сети и топологии;
2. HostManager — приложение, отвечающее за обнаружение конечных устройств;
3. LearningSwitch — приложение, отвечающее за доставку пакетов до пункта назначения.

Для генерации трафика в сети была использована встроенная функция системы Mininet pangall. Замер количества сообщений осуществлялся с помощью инструментов Open vSwitch коммутатора — ovs-ofctl, и его команды snoop. Данная команда была вызвана для всех коммутаторов перед вызовом функции pingall и собирала все OpenFlow сообщения в файлы. Далее по данным файлам считалось общее количество PacketIn и FlowMod сообщений.

На рисунке 5 представлен графики зависимости количества PacketIn сообщений от количества конечных устройств в сети для модифицированной и не модифицированной системы и их

аппроксимирующие функции. Из данного графика видно, что удалось снизить показатель функции на 0.6

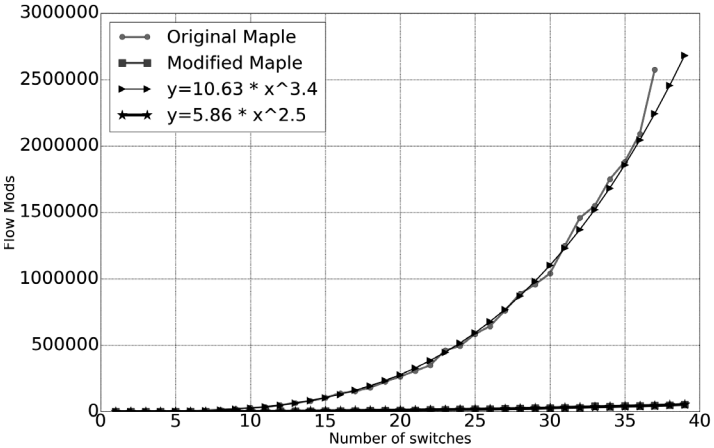

Рисунок 6: График зависимости количества FlowMod сообщений от количества конечных устройств в сети

На рисунке 6 представлен графики зависимости количества FlowMod сообщений от количества конечных устройств в сети для модифицированной и не модифицированной системы и их аппроксимирующие функции. Из данного графика видно, что удалось снизить показатель функции на 0.9 и вдвое уменьшить ее множитель.

5. Заключение

В данной работе разработана эффективная система автоматической генерации правил, которая предоставляет высокоуровневый интерфейс для задания сетевых политик. Данная система построена на базе системы Maple и устраняет ее недостаток, связанный с постоянным обновлением таблицы потоков. Также данная система предоставляет возможность задавать политики для целой сети и задавать специфичные политики для отдельных коммутаторов с помощью высокоуровневых абстракций.

Результатом работы является уменьшение количества системных сообщений на 90%. Это является полезным, потому что в ПКС сетях канал между коммутатором и контроллером может быть медленным, или контроллер может быть вынесен в удаленно расположенный центр обработки данных. Также уменьшение

количества системных сообщений уменьшает нагрузку на процессор потока управления коммутатора, который в современных реалиях является узким местом программно-конфигурируемых сетей.

Литература

1. Voellmy A., Kim H., Feamster N. Procera: a language for high-level reactive network control //Proceedings of the first workshop on Hot topics in software defined networks. – ACM, 2012. – С. 43-48.
2. Reich J. et al. Modular sdn programming with pyretic //Technical Reprot of USENIX. – 2013.
3. Voellmy A. et al. Maple: Simplifying SDN programming using algorithmic policies //ACM SIGCOMM Computer Communication Review. – 2013. – Т. 43. – №. 4. – С. 87-98.
4. McKeown N. et al. OpenFlow: enabling innovation in campus networks //ACM SIGCOMM Computer Communication Review. – 2008. – Т. 38. – №. 2. – С. 69-74.
5. Shalimov A. et al. The Runos OpenFlow Controller //Software Defined Networks (EWSDN), 2015 Fourth European Workshop on. – IEEE, 2015. – С. 103-104.
6. Швецов Д.А., Шалимов А.В., Курсовая работа «Разработка оптимизированного алгоритма автоматической генерации правил в ПКС» ВМК МГУ, 2016. (http://bit.ly/2pU4MQB)

Шемякин Р.О., Петров И.С.

Обеспечение контроля доступа приложений к ресурсам контроллера программно конфигурируемых сетей

Введение

В последние годы парадигма программно-конфигурируемых сетей (далее ПКС) привлекла много внимания со стороны промышленности и в академических кругах [1, 2]. Основной архитектурной особенностью ПКС является передача логики управления сетью на отдельный сетевой элемент – контроллер ПКС или сетевую операционную систему. Контролер ПКС предоставляет сервисы по управлению программируемой сетевой инфраструктурой через открытые интерфейсы. В рамках контроллера работают ПКС-приложения, которые используют сервисы контроллера для реализации логики управления сетью.

Для того чтобы реализовывать свою логику, приложения получают доступ к различным операциям, в том числе и критическим. Как и любое программное обеспечение, приложения могут содержать ошибки, которые могут привести к незапланированному поведению приложения. Более того, использование сторонних приложений может привести к угрозе запуска вредоносной программы с правами управления сетью.

Таким образом, контроллеры должны обеспечивать изоляцию приложений между собой. То есть контроллер должен иметь возможность разделять приложения, работающие на нём, обеспечивать отсутствие взаимного влияния приложений и несанкционированного влияния на работу самого контроллера. Изолированную среду выполнения приложений, которая не допускает взаимодействия с внешней средой, принято называть “песочницей” (англ. sandbox) [5, 6, 7, 9].

Контроллеры также должны обеспечивать управление доступом приложений к интерфейсам управления сетевым оборудованием. Задача обеспечения контроля доступа разделяется на взаимно зависящие процессы управления приложениями: аутентификация, авторизация и учет. Процессы аутентификации ограничивают множество запускаемых приложений, а процесс авторизации и учета устанавливает для приложений ограничения по управлению сетевыми устройствами.

В данной статье проведен сравнительный анализ существующих решений по обеспечению контроля доступа приложений, определены их преимущества и недостатки, обнаружены критические проблемы системы, описанной в [8]. Также в данной статье описывается

разработанная система контроля доступа приложений программно-конфигурируемых сетей к интерфейсам управления сетевыми устройствами, исправляющая критические проблемы [8], которая предоставляет инструмент по заданию политик доступа и механизм, обеспечивающий обязательное выполнение установленных политик доступа.

Структура данной статьи следующая. В разделе 1 дается подробное описание существующих решений по обеспечению контроля доступа приложений и проводится их сравнительный анализ. В разделе 2 описываются принципы построения архитектуры разработанной системы контроля доступа, которые позволяют преодолеть недостатки существующих решений. Раздел 3 содержит описание составных частей разработанной системы контроля доступа. Раздел 4 описывает экспериментальное исследование разработанной системы.

1. Обзор существующих решений

Данный раздел посвящен сравнению существующих решений по обеспечению контроля доступа приложений к интерфейсам управления сетевыми устройствами на контроллере. Целью данного обзора является анализ подходов к организации и реализации контроля доступа, выявление их преимуществ и недостатков и ограничений их применимости.

SDNShield [7] является программным продуктом, предназначенным для встраивания в контроллеры на платформе Java. Изоляция приложений обеспечивается возможностями Java Security Manager [15] и OSGi Conditional Permission Admin service [16].

Политики безопасности описываются в форме разрешения или запрета в доступе к каким-либо ресурсам системы. Политика может быть достаточно гранулярной. Например, может быть задана политика, разрешающая доступ к части файловой системы или только к некоторому набору портов для установления соединений.

ROSEMARY [5] – это ПКС контроллер, ориентированный на обеспечение безопасного исполнения приложений в рамках определяемой политики доступа к ресурсам. Основным компонентом ROSEMARY является монитор ресурсов (Resource Monitor). Этот компонент следит за соблюдением политики использования ресурсов приложениями.

Политика использования ресурсов задается таблицей, где для каждого приложения устанавливаются мягкие и жесткие ограничения использования определенного ресурса. Монитор ресурсов следит за преодолением приложениями установленных ограничений.

AXE [8] является программным продуктом для обеспечения выполнения политик управления сетевыми устройствами и реализующим функции монитора безопасности.

Задача данного монитора безопасности заключается в передаче между контроллером и коммутаторами только тех управляющих сообщений, которые разрешены политиками доступа. Авторы предлагают запускать контроллер, приложения и монитор на одном устройстве. Для контроллера и приложений требуется создать виртуальное окружение. В реализации используется механизм namespaces ядра Linux, позволяющий создать отдельный виртуальный сетевой стек, отсоединить контроллер от сети и настроить связь между контроллером и монитором.

Согласно описаниям, все три программных продукта решают поставленную задачу, используя различные архитектурные решения и технологии. В таблице 1 приведены результаты сравнительного анализа.

Критерии сравнения	**SDNShield**	**ROSEMARY**	**AXE**
Контроль доступа	+	+	–
Изоляция приложений	+	+	–
Задание гранулярных политик доступа	+	–	+
Независимость от контроллера	+	–	+
Независимость от протокола	–	+	+
Применимость в иерархиях контроллеров	–	–	+

Таблица 1. *Результаты сравнительного анализа*

Однако в AXE реализация контроля доступа обладает двумя критическими недостатками.

Первый недостаток заключается в том, что контроль доступа реализован только между контроллером и сетевым оборудованием. Авторы описывают работу монитора безопасности как сетевого экрана, который пропускает или сбрасывает управляющие OpenFlow [3] сообщения. В политиках доступа авторы предлагают указывать имя приложения, для которого политики должны быть применены. При этом отсутствует этап согласованного присвоения имён приложениям и механизм, который бы заставлял контроллер каким-либо образом помечать управляющие сообщения именами приложений.

Второй недостаток заключается в отсутствии изоляции между приложениями на контроллере. Если предположить, что монитор безопасности получает информацию о приложении-авторе, то вредоносное приложение может посылать управляющие сообщения от имени любого приложения, используя его права в системе.

Таким образом, из описания монитора безопасности AXE следует, что он не удовлетворяет первому критерию. Однако, предложенный монитор безопасности может быть использован для контроля доступа всего контроллера с приложениями к сетевым устройствам.

2. Предлагаемая архитектура системы

Для получения преимущества перед существующими решениями было предложено использовать архитектуру монитора безопасности AXE и исправить существующие в ней недостатки.

Контроллер, защищаемый монитором безопасности, реализует следующую функциональность:

1. Обеспечивает изоляцию приложений.
2. Присваивает каждому приложению идентификатор, согласованный с монитором безопасности.
3. Для каждого исходящего управляющего сообщения предоставляет идентификатор приложения-источника монитору безопасности.
4. Доставляет каждое входящее управляющее сообщение только приложениям-адресатам, определённым монитором безопасности.

Монитор безопасности должен дополнительно предоставлять контроллеру список идентификаторов приложений-адресатов пересылаемого управляющего сообщения.

На рисунке 1 изображена архитектура предлагаемого решения. Различными типами стрелок обозначены различные способы передачи данных. Каждое приложение и контроллер находятся в своем виртуальном окружении и не имеют прямого доступа к сетевым устройствам.

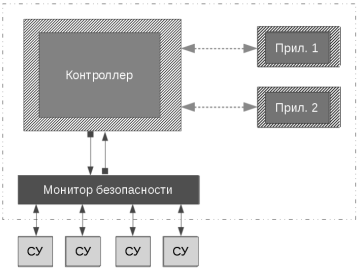

Рисунок 1. *Архитектура предлагаемого решения*

В предложенной архитектуре монитор безопасности является единственным местом, в котором осуществляется контроль доступа. Это означает, что такой монитор может быть применен для любого контроллера, удовлетворяющего описанным выше требованиям.

3. Реализация

Разработанная система контроля доступа состоит из следующих компонентов: Монитор безопасности, Система запуска, Альтернативная библиотека программных интерфейсов контроллера.

В качестве базового контроллера был выбран Ryu [14]. Данный контроллер написан полностью на языке Python, обладает монолитной архитектурой (ядро и приложения работают в рамках одного процесса). Ryu реализует событийно-ориентированную модель взаимодействия с приложениями. Ядро контроллера взаимодействует с приложениями через индивидуальные очереди сообщений. Работа приложений заключается в обработке сообщений о событиях в сети и составлении запросов по управлению сетевыми устройствами. Таким образом, в Ryu приложения хорошо отделимы от ядра, что позволяет реализовать их полную изоляцию, не нарушая общую архитектуру контроллера.

3.1. Изоляции приложений

Контроллер Ryu реализует событийно-ориентированную модель взаимодействия с приложениями. Приложения подписываются на интересующие их события и реализуют функции-обработчики этих событий. У каждого приложения есть очередь событий, которую оно обрабатывает.

В ядро контроллера были внесены изменения, согласно которым входящие события отправляются только тем приложениям, которые на них подписаны, и для которых было получено одобрение от монитора безопасности. Все запросы от приложений ядро контроллера помечает идентификатором приложения-отправителя и направляет на монитор безопасности.

Была разработана альтернативная библиотека программных интерфейсов Ryu. Эта библиотека позволяет запускать существующие Ryu-приложения отдельно от ядра контроллера. Взаимодействие с ядром Ryu происходит через модифицированные очереди сообщений, в которых для передачи сообщений используются сокеты домена Unix.

3.2. Система запуска

Система запуска создаёт для ядра контроллера и для каждого приложения собственное виртуальное окружение, устанавливает связи между ними и подключает ядро контроллера к монитору безопасности.

Для организации виртуального окружения были выбраны следующие механизмы [4]: система разграничения прав доступа к файлам (POSIX ACL) и механизм пространств имён.

Система разграничения прав доступа к файлам предоставляет интерфейс для создания виртуального окружения в виде рабочего пространства пользователя в системе, к которому ограничен доступ других пользователей. Процесс, запущенный от имени некоторого

пользователя будет ограничен в доступе к ресурсам системы правами этого пользователя. Запуск ядра контроллера и приложений от имени разных пользователей позволяет использовать этот механизм для их изоляции.

Механизм пространств имён позволяет изолировать ресурсы системы для определённой группы процессов. Этот механизм может быть применён для установления ограничений на использование сетевых интерфейсов вплоть до полного запрета. Для этого достаточно определить пространство имён типа *network*, в котором не будет ни одного сетевого интерфейса. Запустить любую программу в таком пространстве имён можно с помощью стандартной утилиты *unshare*. Такая программа будет полностью лишена возможности сетевого взаимодействия.

Алгоритм работы системы запуска:

1. Создаётся пользователь *KernelUser*, от имени которого будет запущено ядро контроллера, и группу *KernelGroup*;
2. Создаются два Unix сокета доступных на чтение и запись членам *KernelGroup* – один для передачи сообщений от ядра к монитору безопасности, второй – наоборот;
3. Запускается монитор безопасности;
4. Инициализируется каждое приложение по следующему алгоритму:
    a. Создаётся пользователь *AppUser* и группа *AppGroup* на основе идентификатора приложения;
    b. Система развертывания распаковывает в домашней папке *AppUser* код приложения и код альтернативной библиотеки программных интерфейсов;
    c. Создаются два Unix сокета доступных на чтение и запись членам *AppGroup* – один для передачи сообщений от ядра к приложению, второй – наоборот;
    d. Запускается приложение в пустом сетевом пространстве имён;
    e. Добавляется пользователь *KernelUser*, от имени которого будет запущено ядро контроллера, в *AppGroup*;
5. Запускается ядро контроллера в пустом сетевом пространстве имен.

На рисунке 2 показаны ядро контроллера, два приложения и монитор безопасности, запущенные на одном устройстве как отдельные процессы. Процессы с пунктирной границей не имеют возможности осуществлять сетевое взаимодействие. Стрелками обозначены Unix-сокеты, связывающие различные процессы.

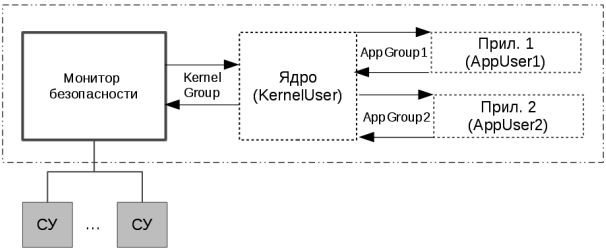

Рисунок 2. Организация виртуального окружения для ядра и приложений

3.3. Монитор безопасности

Разработанный монитор безопасности устроен аналогично монитору AXE. Обработчик запросов реализован на языке C++. Разбор политик и построение дерева фильтрации реализованы на языке Python.

3.3.1. Разбор политик

На вход монитор принимает конфигурационный файл с политиками в формате YAML. В рамках одной политики могут быть заданы правила обработки для нескольких приложений. Для каждого приложения должны быть заданы разрешенные для него типы управляющих сообщений с правилами их обработки.

Правила обработки определяются двумя значениями. Значение *filters* (условия) задает список условий, которым должно удовлетворять управляющее сообщение. Значение *decision* (решение) определяет действие по отношению к управляющему сообщению: передавать дальше или сбрасывать (*permit* или *deny*).

3.3.2. Построение дерева фильтрации

На множестве полей, используемых в правилах, установлен частичный порядок. Также в рамках одного заголовка поля упорядочены по их месту в заголовке. Установленный порядок позволяет построить процесс фильтрации управляющих сообщений по направлению от заголовков верхнего уровня к заголовкам более низких уровней, что активно используется при построении дерева фильтрации.

Дерево фильтрации определяет порядок проверки полей анализируемого управляющего сообщения. В каждом нелистовом узле указано поле, а ребра, исходящие из этого узла, помечены условиями на это поле. В случае выполнения одного из условий совершается переход по соответствующему ребру. У каждого узла есть особое ребро, по которому переход совершается в том случае, если условия на других ребрах не были выполнены. В каждой листовой вершине находятся правила, условия которых не противоречат условиям на пути от корня дерева до этой листовой вершины. Таким образом, спуск по дереву

фильтрации позволяет определить подмножество правил, которые выполняются на анализируемом сообщении.

Дерево фильтрации строится по группе правил обработки рекурсивно. Все поля, на которые накладываются условия в группе правил, объединяются и упорядочиваются согласно описанному выше правилу и выбирается первое из них. Далее, происходит построение узла дерева фильтрации по выбранному полю.

Если поле обычное, то все условия на точное совпадение заменяются на условия совпадения из одноэлементного отрезка. Получившееся множество отрезков подвергается дроблению таким образом, чтобы отрезки не пересекались и в объединении давали множество, равное объединению исходных отрезков. Далее, создаётся узел фильтрации по рассматриваемому полю и для каждого отрезка из полученного множества создается ребро, помеченное условием равенства одному значению из отрезка. Если поле маскированное, то создаётся узел фильтрации по рассматриваемому полю и для каждого значения из условий создается ребро, помеченное этим условием.

После построения узла фильтрации для каждого ребра определяется подмножество правил из группы, которые не противоречат условию на ребре. Если в правиле отсутствует условие на рассматриваемое поле, то это правило не противоречит ни одному условию на ребрах, в том числе и на особом ребре с условием по-умолчанию. Когда множества правил для ребер определены, процедура построения дерева фильтрации запускается рекурсивно для каждого такого множества. Если множество правил для ребра пусто, то ребро соединяется со специальным листовым узлом с пустым множеством сработавших правил. Если в группе правил для всех полей из условий уже созданы узлы фильтрации, то создаётся или используется уже существующая листовая вершина, в которой сработали все правила из группы.

3.3.3. Особенности группировки правил обработки

Правила обработки разделяются на два множества, в зависимости от их естественного направления передачи. Например, *PacketIn* [3] проверяются только по направлению к контроллеру, а *PacketOut* – только от контроллера.

Правила для сообщений по направлению от контроллера к сетевым устройствам группируются по типу управляющего сообщения и идентификатору приложения. Для принятия решения монитору необходимо применить правила только одной из выделенных групп. Для сообщений от контроллера монитор безопасности по типу управляющего сообщения и идентификатору приложения выбирает нужное дерево фильтрации, спускается по нему и в листовой вершине принимает решение на основе данного узла.

Правила для сообщений по направлению от сетевых устройств к контроллеру группируются только по типу управляющего сообщения. Такая группировка обусловлена тем, что управляющее сообщение может предназначаться нескольким приложениям. Для сообщений от сетевых устройств монитор безопасности выбирает нужное дерево фильтрации только по типу управляющего сообщения, спускается по нему и в листовой вершине узнаёт список идентификаторов приложений адресатов и добавляет эту информацию в сообщения для контроллера.

Таким образом, при инициализации монитора безопасности строятся отдельные структуры для двух направлений передачи управляющих сообщений.

3.4. Взаимодействие между монитором и контроллером

В рассматриваемой архитектуре контроллер всегда должен сопровождать управляющее сообщение единственным идентификатором приложения-источника, а монитор безопасности – списком идентификаторов приложений-адресатов. Поэтому при передаче управляющих сообщений контроллер и монитор будут добавлять разные заголовки.

Контроллер добавляет заголовок размером 4 байта: первые два байта составляют так называемый *magic number*, вторые – идентификатор приложения.

Монитор добавляет заголовок переменного размера: первые два байта составляют *magic number*, вторые – количество передаваемых идентификаторов приложений. Далее, следуют идентификаторы приложений по 2 байта каждый. Если идентификаторов было нечетное количество, то дописывается идентификатор 0x0000, который будет проигнорирован контроллером.

4. Экспериментальное исследование

Тестирование проводилось на следующей конфигурации оборудования:

- *ОС*: Ubuntu Server 16.04 xenial
- *Ядро*: x86_64 Linux 4.8.0-51-generic
- *CPU*: Intel Core i7-6600U CPU @ 3.4GHz
- *RAM*: 16Gb

Были проведены тесты с нагрузкой от 1000 до 10000 сообщений *PacketIn* в секунду, и проводился подсчет отправленных в ответ *PacketOut* сообщений. На рисунке 3 изображены показатели пропускной способности контроллера. Светлым обозначены результаты оригинального контроллера, темным – модифицированного. По результатам тестирования видно, что система контроля доступа снижает пропускную способность контроллера от *1%* на 1000 *PacketIn/сек*, до *14%* на 10000 *PacketIn/сек*. Сообщения в контроллере испытывают

дополнительные задержки, связанные с их упаковкой, распаковкой и передачей через Unix-сокеты. Кроме того, очередь исходящих сообщений с контроллера, расположенная в ядре, является узким местом системы.

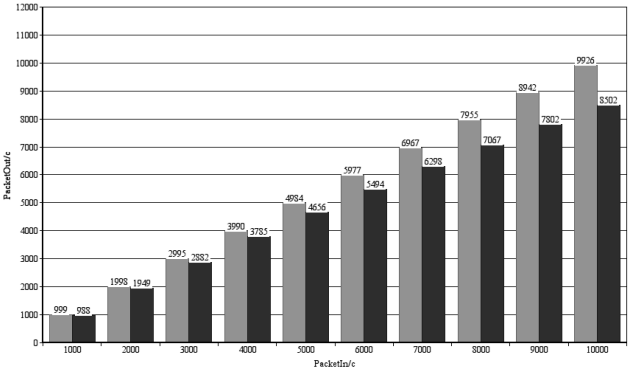

Рисунок 3. *Оценка пропускной способности системы*

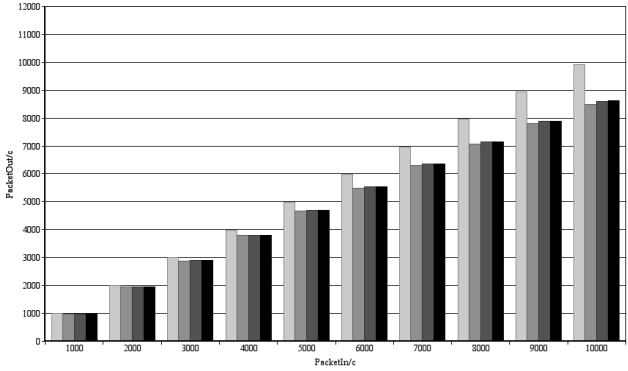

Рисунок 4. *Зависимость пропускной способности от количества потоков*

Для того чтобы определить насколько сильно влияет монитор безопасности на время обработки пакета в системе, были проведены аналогичные тесты с изменением количества потоков на мониторе безопасности от 1 до 10. Результаты тестирования показали прирост не более *1.5%* пропускной способности. На рисунке 4 изображены результаты тестов: светлым обозначены результаты оригинального контроллера, более темным – модифицированного (слева направо 1, 5 и 10 потоков).

Из результатов тестирования был сделан вывод, что снижение задержки на мониторе безопасности незначительно по сравнению с

абсолютной величиной задержки. Для корректной оценки пропускной способности монитора безопасности были проведены тесты в отсутствие контроллера с количеством потоков от 1 до 12. Тесты проводились с использованием двух конфигураций политик доступа из 100 и 100000 правил. Вторая конфигурация была подобрана так, чтобы в её дереве фильтрации было много “тяжелых” узлов – с большим количеством рёбер. Результаты тестирования отражены на рисунке 5. По ним можно сделать вывод, что дополнительная задержка при спуске по “тяжелому” дереву фильтрации не превосходит 40 мкс. В целом, пропускная способность монитора безопасности как минимум в 1.5-2 раза меньше, чем у современных контроллеров [10, 11, 12].

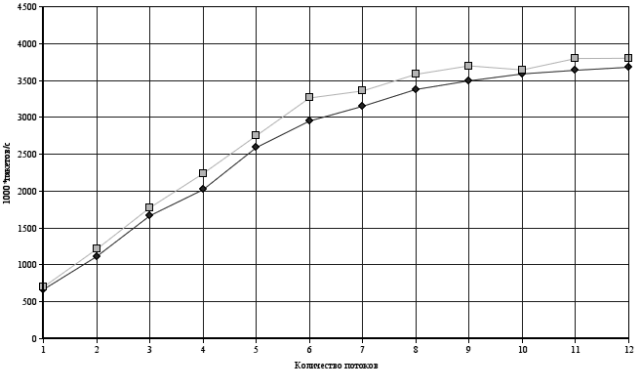

Рисунок 5. *Оценка пропускной способности монитора безопасности*

5. Заключение

В статье приведен сравнительный анализ существующих решений по обеспечению контроля доступа приложений к интерфейсам управления сетевыми устройствами на SDN контроллере, определены их преимущества и недостатки, обнаружены критические проблемы в [8]. Описана разработанная система задания и применения политик доступа, исправляющая критические проблемы [8]. Описана реализация прототипа системы контроля доступа к интерфейсам управления сетевыми устройствами и ее экспериментальное исследование.

Литература

1. Смелянский Р.Л. Программно-конфигурируемые сети // «Открытые системы», No9, 2012 [Электронный ресурс] URL: https://www.osp.ru/os/2012/09/13032491/ (дата посещения: 14.05.2017)

2. S. Scott-Hayward, G. O'Callaghan, and S. Sezer, SDN Security: A Survey // IEEE SDN for Future Networks and Services (SDN4FNS), 2013, pp. 1–7.
3. OpenFlow - Open Networking Foundation [Электронный ресурс] URL: https://www.opennetworking.org/sdn-resources/openflow (дата посещения: 14.05.2017)
4. Zhai G., Li Y. Analysis and Study of Security Mechanisms inside Linux Kernel // Security Technology, 2008. SECTECH'08. International Conference on. – IEEE, 2008. – С. 58-61.
5. Shin S. et al. Rosemary: A robust, secure, and high-performance network operating system //Proceedings of the 2014 ACM SIGSAC conference on computer and communications security. – ACM, 2014. – С. 78-89.
6. Röpke C., Holz T. On network operating system security //International Journal of Network Management. – 2016. – Т. 26. – №. 1. – С. 6-24.
7. Wen X. et al. SDNShield: Reconciliating Configurable Application Permissions for SDN App Markets // Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on. – IEEE, 2016. – С. 121-132.
8. Yu D. Access control for network management. – University of Cambridge, Computer Laboratory, 2017. – №. UCAM-CL-TR-898.
9. Porras P. A. et al. Securing the Software Defined Network Control Layer //NDSS. – 2015.
10. Shalimov A. et al. The Runos OpenFlow Controller // Software Defined Networks (EWSDN), 2015 Fourth European Workshop on. – IEEE, 2015. – С. 103-104.
11. OpenDaylight: A Linux Foundation Collaborative Project. [Электронный ресурс]. URL: http://www.opendaylight.org (дата посещения: 14.05.2017)
12. ONOS: Open Network Operating System. [Электронный ресурс] URL: http://onosproject.org/ONOS (дата посещения: 14.05.2017)
13. Floodlight OpenFlow Controller -Project Floodlight [Электронный ресурс] URL: http://www.projectfloodlight.org/floodlight/ (дата посещения: 14.05.2017)
14. Ryu, Component-based Software Defined Networking Framework [Электронный ресурс] URL: https://osrg.github.io/ryu/ (дата посещения: 14.05.2017)
15. Oracle Java Documentation "Default Policy Implementation and Policy File Syntax" [Электронный ресурс] URL: https://docs.oracle.com/javase/8/docs/technotes/
guides/security/PolicyFiles.html (дата посещения: 14.05.2017)
16. OSGi™ Alliance – The Dynamic Module System for Java [Электронный ресурс] URL: https://www.osgi.org/ (дата посещения: 14.05.2017)

Шендяпин А.С., Петров И.С.

Исследование методов проведения атаки Man-in-the-Middle в программно-конфигурируемых сетях

Введение

Программно-конфигурируемые сети SDN [1] являются новым подходом к построению компьютерных сетей, нацеленным на решение научно-технических, экономических проблем, а также проблем развития современных компьютерных сетей.

Идея таких сетей была сформулирована специалистами университетов Стэнфорда и Беркли еще в 2006 году, а инициированные ими исследования нашли поддержку не только в академических кругах, но и были активно восприняты ведущими производителями сетевого оборудования, образовавшими в марте 2011 года консорциум Open Networking Foundation (ONF). Его учредителями выступили Google, Deutsche Telekom, Facebook, Microsoft, Verizon и Yahoo. Состав ONF быстро расширяется, в нее уже вошли такие компании, как Brocade, Citrix, Oracle, Dell, Ericsson, HP, IBM, Marvell, NEC и ряд других. Одной из первых практическую реализацию SDN предложила компания Nicira, вошедшая недавно в состав VMware.

В SDN уровни управления сетью (control plane) и передачи данных (data plane) разделяются за счет переноса функций управления (маршрутизаторами, коммутаторами и т.п.) в приложения, работающие на отдельном сервере (контроллере).

Эта особенность позволила решить многие открытые проблемы, такие как сокрытие деталей уровня передачи данных, предоставление удобного интерфейса для написания сетевых приложений, реализация более сложных алгоритмов маршрутизации и многое другое.

Еще одно отличие программно-конфигурируемых сетей – управление потоками, а не отдельными пакетами [4]. Под управлением потоками понимается возможность задания правил, по которым производится маршрутизация всех пакетов с одинаковым заголовком. За установку соответствующих правил отвечает SDN контроллер, к которому подключены SDN коммутаторы, осуществляющие маршрутизацию потоков в сети.

Наиболее широкое распространение среди протоколов взаимодействия контроллера с сетевыми устройствами получил протокол OpenFlow [2]. Согласно данному протоколу, каждый SDN коммутатор должен содержать одну или более таблиц потоков и поддерживать канал для связи с удаленным контроллером – сервером.

Каждая таблица потоков в коммутаторе содержит набор записей об обрабатываемых данным коммутатором потоках. Каждая такая запись состоит из полей-признаков, счетчиков и набора инструкций. OpenFlow правило описывает набор действий, производимых над пакетами, принадлежащими обрабатываемому данным правилом потоку.

В данной статье проводится исследование уязвимости программно-конфигурируемых сетей к атакам Man-in-the-middle, реализуемых с использованием скомпрометированного коммутатора. Реализуются различные тестовые сценарии данных атак в виртуальной сети с использованием Mininet [5] под управлением SDN контроллера Ryu [6].

Структура данной статьи следующая. В разделе 1 описываются различные угрозы безопасности сети при захвате сетевого оборудования. Раздел 2 содержит описание основных атак, проводимых на уровень передачи данных. В разделе 3 описаны методы проведения атаки Man-in-the-Middle в SDN. В разделе 4 проведено экспериментальное исследование реализации рассмотренных методов атак Man-in-the-Middle, использующих скомпрометированный коммутатор.

1. Угроза захвата сетевого оборудования

Скомпрометированным коммутатором будем называть коммутатор из сети, управляемой SDN контроллером, который в действительности находится под контролем злоумышленника.

Возможные сценарии захвата коммутатора перечислены ниже:

1) Если SDN-контроллер захвачен некоторым злоумышленником, то скомпрометированными оказываются все SDN коммутаторы, так как атакующий может использовать функции контроллера для управления всей SDN сетью.
2) Кроме захвата SDN контроллера, злоумышленник может произвести захват канала управления между контроллером и некоторым коммутатором (рис. 1). Злоумышленник, используя канал управления, может перенастроить коммутатор так, чтобы он подчинялся контроллеру атакующего. Этот тип атак был назван “Control-channel Hijacking” [8].
3) Как и в любом программном и аппаратном обеспечении, в SDN коммутаторе могут присутствовать уязвимости, эксплуатация которых некоторым атакующим, может привести к захвату (компрометации) коммутатора, содержащего такие уязвимости.

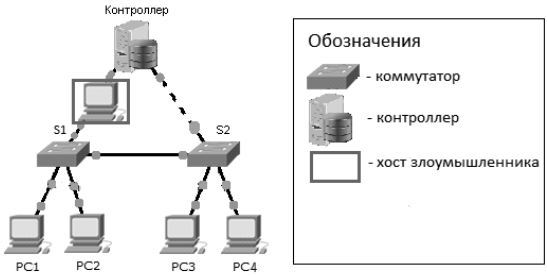

Рисунок 1. *Захват канала управления [8]*

Основной атакой, которую может произвести атакующий, используя скомпрометированный коммутатор, является атака типа Man-in-the-Middle (MitM) [9]. Данная атака заключается в том, что атакующий может перехватывать сетевой трафик, проходящий через скомпрометированный коммутатор.

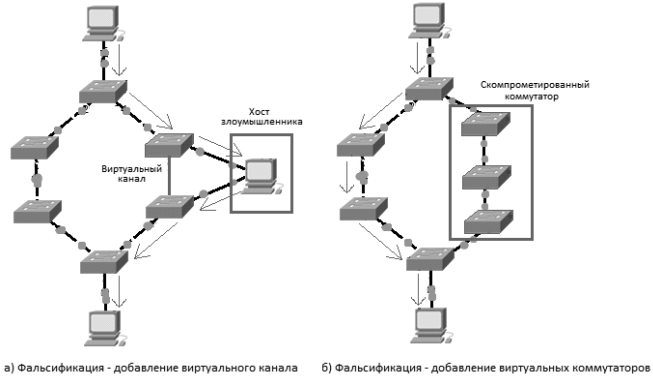

Рисунок 2. *Предоставление контроллеру ложной информации [8]*

Кроме идущего непосредственно через скомпрометированный коммутатор трафика, атакующий может влиять и на окружение коммутатора в сети. Например, коммутатор может сообщить контроллеру о несуществующем канале между ним и другим коммутатором (рис. 2, (а)), что приведет к увеличению числа потоков, направляемых контроллером через данный коммутатор, что, в свою очередь, предоставит злоумышленнику большее количество трафика для анализа.

Другая ситуация заключается в том, что злоумышленник с помощью скомпрометированного коммутатора добавляет в сеть виртуальные коммутаторы, таким образом удлиняя пути, проходящие через данный коммутатор (рис. 2, (б)). Это приводит к тому, что часть

трафика будет перенаправлена на другие каналы, что увеличит их загрузку и может вызвать перегрузку в сети [8].

В данной статье будут рассматриваться методы проведения атаки Man-in-the-Middle, нацеленной только на уровень передачи данных.

2. Атаки на уровень передачи данных

Все атаки на уровень передачи данных можно разделить на активные и пассивные.

Активными атаками называются атаки, изменяющие существующие, либо генерирующие новые потоки трафика. К таким атакам, например, относятся: DDoS-атака, изменение и сброс данных, передаваемых по каналу, проходящему через скомпрометированный коммутатор и перераспределение трафика по портам, вызывающее перегрузку на определенных портах. Данные атаки могут приводить к серьезным задержкам в работе сети, перегрузкам и сбросу пакетов, а также нарушению конфиденциальности и целостности пользовательских данных, передаваемых через сеть. Подобные атаки проводятся атакующим при помощи модификации потоковой таблицы скомпрометированного коммутатора.

Методы противодействия активным атакам – хранение информации о потоковых таблицах коммутаторов, мощностях их портов и информации о сетевых каналах на контроллере, а также сравнение этой информации с данными, получаемыми в результате мониторинга сети. Мониторинг обычно производится выборочно – проверкой соблюдения случайно выбранных правил на случайно выбранных коммутаторах [9].

Злоумышленник может изменить функционал коммутатора таким образом, чтобы он мог предоставлять контроллеру недостоверную информацию о состоянии потоковой таблицы. При этом контроллер, проводя анализ ответов коммутаторов, не сможет обнаружить изменение потоковой таблицы и, таким образом не сможет обнаружить скомпрометированный коммутатор.

В таких случаях можно использовать информацию, предоставляемую пользователями сети, так как изменения таблицы потоков зачастую сопровождаются заметными задержками при передаче данных. Поэтому возможно использование специальных серверов, на которые пользователи будут присылать свои подозрения о захватах каналов, после чего контроллер будет проверять их [10].

Пассивными атаками называются атаки, нацеленные на прослушивание каналов связи и анализ полученного трафика на самом скомпрометированном коммутаторе. Данные атаки намного сложнее

обнаружить, так как в состоянии сети не происходит никаких изменений.

3. Методы проведения атак

В данной статье рассматриваются методы проведения активной атаки Man-in-the-Middle на уровень передачи данных в SDN сети с использованием скомпрометированного коммутатора. Далее предполагается, что контроллер не проводит мониторинга сети. В данной статье не рассматриваются методы проведения атак, использующие несколько скомпрометированных коммутаторов, поэтому предполагается, что злоумышленник скомпрометировал единственный SDN коммутатор.

Рассмотрим различные сценарии, которые могут возникнуть при проведении атаки Man-in-the-Middle в SDN сети с произвольной топологией.

Рисунок 3. *Сценарий 1 – непосредственное подключение к коммутатору*

Первый сценарий заключается в том, что злоумышленник подключен непосредственно к скомпрометированному коммутатору, и хочет прослушивать идущий через него поток (рис. 3). В этом случае методом проведения атаки Man-in-the-Middle будет являться изменение потоковой таблицы коммутатора следующим образом: атакующий устанавливает на коммутатор правила, дублирующие входящий поток на порт, соединенный с хостом атакующего. Таким образом, одна копия отправляется атакующему, а другая дальше по назначению, чтобы контроллер и атакуемый хост не обнаружили факт ее проведения.

Однако такой метод неприменим, если атакующий не связан с коммутатором непосредственно. Если между злоумышленником и скомпрометированным коммутатором находятся другие коммутаторы, дублируемые пакеты не пройдут через них, а будут отправлены на контроллер, так как промежуточных коммутаторах могут быть не установлены правила обработки пакетов с данными заголовками.

Следующий сценарий заключается в том, что между скомпрометированным коммутатором и хостом злоумышленника могут находиться нескомпрометированные коммутаторы (рис. 4). Атакующий хочет прослушивать канал между атакуемым хостом и общедоступным

сетевым источником (например, сервером), проходящий через скомпрометированный коммутатор. Однако если на промежуточных коммутаторах не установлены соответствующие правила коммутации, то дублированные пакеты будут отправлены на контроллер, и тот сможет обнаружить атаку.

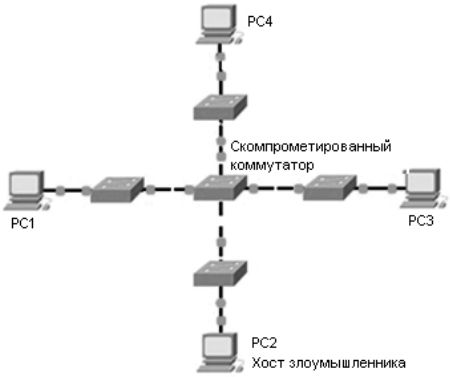

Рисунок 4. *Сценарий 2 – подключение через промежуточные коммутаторы.*

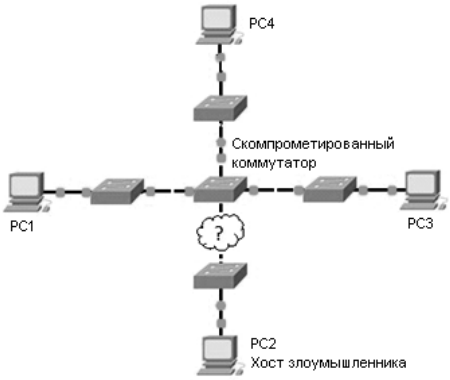

Рисунок 5. *Сценарий 3 – отсутствие информации о топологии сети*

Метод проведения атаки Man-in-the-Middle при наличии промежуточных коммутаторов заключается в следующем. Прежде чем дублировать трафик, злоумышленнику необходимо установить соединение с хостом, на котором работает сервер. Данное соединение должно пройти через скомпрометированный коммутатор. Тогда на коммутаторах, находящихся между скомпрометированным коммутатором и хостом злоумышленника, соответствующие правила будут установлены контроллером.

Далее атакующий устанавливает на скомпрометированный коммутатор правила, которые изменяют заголовки пакетов из дублированного потока таким образом, чтобы данные пакеты были корректно обработаны на промежуточных коммутаторах. Для этого в качестве адреса отправителя указывается адрес сервера, а в качестве адреса получателя – адрес атакующего. Таким образом, происходит инкапсуляция дублированного потока в поток общения с сервером, установленным ранее.

Тем не менее, предположение о том, что канал пойдет так, как это нужно атакующему, а так же, что сам атакующий знает об этом, является нереалистичным. Поэтому третий сценарий заключается в том, что атакующий не соединен напрямую со скомпрометированным коммутатором и ему неизвестна топология сети, но у него есть только список ip-адресов L_{IP} данной сети (рис. 5).

В данном случае метод проведения атаки Man-in-the-Middle заключается в следующем. Прежде, чем проводить атаку, злоумышленнику необходимо обнаружить возможный поток для сокрытия дублируемого трафика. Для обнаружения данного потока, атакующему необходимо провести сканирование сети, представленное в алгоритме 1.

В описании алгоритма 1 используются следующие обозначения:

- L_{IP} – список IP адресов сети,
- S – скомпрометированный коммутатор,
- T_S – потоковая таблица коммутатора S,
- $ip_{attacker}$ – IP адрес атакующего.

1:	**begin**
2:	**for** *ip* **in** L_{IP}:
3:	**begin**
4:	Установить соединение с хостом по адресу *ip*;
5:	**for** *rule* **in** T_S:
6:	**if** $rule.ip_{src} = ip$ ***and*** $rule.ip_{dst} = ip_{attacker}$ ***then***
7:	Установить правило, изменяющее заголовки дублированного потока на $rule.ip_{src}$ и $rule.ip_{dst}$;
8:	**return**;
9:	**end**
10:	**end**

Алгоритм 1: *Атака MitM, использующая сканирование сети*

В данном алгоритме происходит сканирование сети при помощи подключения к различным хостам сети, с целью установления потока, проходящего через скомпрометированный коммутатор.

Проверка наличия необходимого потока происходит в строке 6. Установка правил, необходимых для проведения атаки Man-in-the-Middle происходит в строке 7. Таким образом, когда поток найден, дублированный поток будет корректно обработан на промежуточных коммутаторах и доставлен на хост атакующего.

4. Реализация и экспериментальное исследование

Описанные в предыдущей главе атаки, были реализованы в виде программы на языке Python. Следует отметить, что реализованная программа позволяет провести значительно большее число атак, такие как некорректная коммутация, flood-атаки, drop-атаки, манипулирование пакетами, MitM-атака.

В качестве данных реализованной программы используется набор ip адресов хостов в сети, а в качестве входных параметров – тип проводимой атаки, а также адреса пользователей, трафик между которыми злоумышленник будет обрабатывать и перехватывать на скомпрометированном коммутаторе.

Программа изменяет правила на скомпрометированном коммутаторе с помощью управляющего *ofctl* канала [7].

Атака некорректной коммутации пакетов достигается за счет сброса пакетов на коммутаторе либо отправке их на неправильные выходные порты. Данные атаки являются частными случаями атак на отказ в обслуживании, для создания дополнительных нагрузок в сети. Атакующий может вызвать подобные нагрузки при помощи перераспределения пакетов по выходным портам так, что коммутаторы, присоединенные к данным портам, перестанут справляться с обработкой пакетов, и в сети будет происходить их сброс.

Также реализованная программа позволяет проводить атаку Man-in-the-Middle, при которой данные, передаваемые получателю, проходят через хост злоумышленника, производящий изменение данных пакетов. В этом случае на скомпрометированном коммутаторе необходимо перенастроить пересылку пакетов так, чтобы они до отправки получателю проходили через хост злоумышленника.

При этом также следует изменять адреса в заголовке пакетов – во избежание отправки пакетов на контроллер и обнаружения атаки. При приходе на коммутатор очередного пакета, его адрес изменяется на адрес хоста злоумышленника. Когда же злоумышленник заканчивает изменение или просмотр данных и отправляет пакет получателю, на коммутаторе адрес атакующего заменяется на адрес отправителя. Таким образом, происходит сокрытие атаки от получателя и отправителя.

Следует отметить, что чем дальше хост злоумышленника находится от скомпрометированного коммутатора, тем проще

обнаружить атаку MitM – из-за резкого увеличения задержки доставки пакетов.

Ниже приведена таблица 1 с действиями над пакетами, установка которых требуется на скомпрометированном коммутаторе для проведения той или иной атаки.

Атака	Действия
Сброс пакетов	`Drop`

Дублирование пакетов	`output:`*`port`*`,`
	`mod_dl_dst:`*`atk_dst`*`,`
	`output:`*`atk_port`*

Неправильная коммутация	`output:`*`another_port`*
Flood-атака	`Flood`
Man-in-the-middle	`mod_dl_dst:`*`atk_dst`*`,` `output:`*`atk_port`* `mod_dl_dst:`*`dst`*`, output:`*`port`*

Таблица 1. *Устанавливаемые OpenFlow правила*

Здесь *atk_dst* – MAC-адрес атакующего, *atk_port* – порт атакующего, а *dst* и *port* – MAC-адрес и порт, которые должны использоваться для правильной коммутации, *another_port* – какой-либо отличный от правильного порт.

Экспериментальное исследование проводилось с помощью эмулятора – Mininet [5]. В качестве OpenFlow-контроллера был использован контроллер Ryu [6], а качестве SDN коммутатора – OpenVSwitch [7]. Для маршрутизации пакетов было использовано приложение l2-router [11], запускаемое на контроллере, которое производит маршрутизацию пакетов, на основе MAC-адресов.

Экспериментальное исследование показало, что атаки Man-in-the-Middle, проводимые с помощью скомпрометированного коммутатора, являются реальной угрозой безопасности программно-конфигурируемых сетей.

5. Заключение

В данной статье было проведено исследование уязвимости программно-конфигурируемых сетей к атакам Man-in-the-Middle при помощи скомпрометированного коммутатора. Были разработаны методы проведения данных атак в различных сетевых сценариях и проведены на виртуальной тестовой топологии.

Исследование показало, что скомпрометированные коммутаторы являются реальной угрозой безопасности программно-конфигурируемых сетей, и необходимы средства для обнаружения таких коммутаторов.

Литература

1. Смелянский Р. Л. Программно-конфигурируемые сети //Открытые системы. СУБД. – 2012. – Т. 9. – С. 23-26.
2. McKeown N. et al. OpenFlow: enabling innovation in campus networks //ACM SIGCOMM Computer Communication Review. – 2008. – Т. 38. – № 2. – p. 69-74.
3. Specification O. F. S. Version 1.3. 2 (Wire Protocol 0x04). – 2013.
4. Kreutz D. et al. Software-defined networking: A comprehensive survey //Proceedings of the IEEE. – 2015. – Т. 103. – № 1. – p. 14-76.
5. Team M. Mininet: An instant virtual network on your laptop (or other PC). – 2012.
6. Ryu SDN Framework Community, "WHAT'S RYU?" 2015.
7. Linux Foundation, Documentation, Open vSwitch. "What is Open vSwitch?".
8. Antikainen M., Aura T., Särelä M. Spook in your network: Attacking an sdn with a compromised openflow switch //Nordic Conference on Secure IT Systems. – Springer International Publishing, 2014. – p. 229-244.
9. Chi P. W. et al. How to detect a compromised SDN switch //Network Softwarization (NetSoft), 2015 1st IEEE Conference on. – IEEE, 2015. – С. 1-6.
10. Du X. et al. Traffic-based Malicious Switch Detection in SDN //International Journal of Security and Its Applications. – 2014. – Т. 8. – № 5. – p. 119-130.
11. Documentation, GitHub – harmjan/l2-router: A proactive scheduling layer 2 router for the Ryu SDN controller.

Костенко В.А., Чупахин А.А.

Влияние миграции на эффективность использования ресурсов центров обработки данных[1]

В работе рассматриваются центры обработки данных (ЦОД) работающие в режиме Infrastructure-as-a-Service (IaaS) [1]. Задача планирования вычислений в ЦОД заключается в отображении запросов на создание виртуальных машин и виртуальных систем хранения данных, соединённых виртуальными каналами, на вычислительные сервера и сервера хранения данных и построении маршрутов для виртуальных каналов в сети обмена ЦОД.

В ходе эксплуатации ЦОД происходит снятие виртуальных ресурсов с выполнения и запуск новых. При попытке разместить новый виртуальный ресурс в ЦОД может возникнуть следующая ситуация. Суммарное количество свободных физических ресурсов по всем запрошенным критериям качества обслуживания [2] хватает для размещения виртуального ресурса, но не существует физического элемента, на который можно было бы разместить этот виртуальный ресурс с выполнением всех запрошенных для него критериев качества обслуживания. То есть в ходе эксплуатации ЦОД возникает фрагментация физических ресурсов, которая приводит к снижению эффективности их использования по критериям загрузка и количество размещенных виртуальных ресурсов. Фрагментация физических ресурсов может устраняться, если допустима миграция работающих виртуальных ресурсов. Миграция виртуального ресурса должна осуществляться без его останова.

В данной работе исследуется эффективность получаемых отображений по критериям загрузка физических ресурсов ЦОД и количество размещенных запросов для случаев, когда допустима и не допустима миграция работающих виртуальных ресурсов.

Алгоритм построения отображения запросов на физические ресурсы центра обработки данных с построением плана миграции работающих виртуальных ресурсов основан на алгоритме, описанном в [3]. Данный алгоритм сочетает жадные стратегии и ограниченный перебор [4]. По жадному критерию выбирается запрос из множества поступивших запросов, и размещаются все его элементы. Если не удалось разместить очередной элемент запроса, то вызывается процедура ограниченного перебора. Для процедуры ограниченного перебора задается параметр, который позволяет регулировать

[1] Работа выполнена при финансовой поддержке РФФИ, грант № 16-07-01237.

вычислительную сложность и точность алгоритма. Процедура построения плана миграции вызывается в том случае, когда процедура ограниченного перебора завершилась успешно. Для всех виртуальных ресурсов, поменявших свое расположение, строится план миграции. Если построенный план миграции удовлетворяет заданному ограничению на время выполнения, то алгоритм переходит к обработке следующего запроса, иначе данный запрос считается неразмещенным.

Принцип работы процедуры ограниченного перебора проиллюстрирован на рис.1.

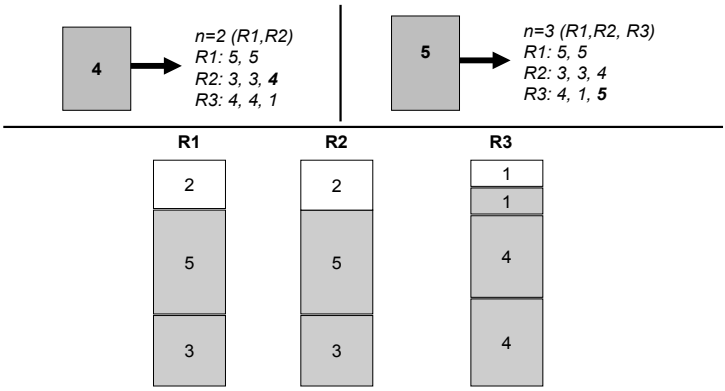

Рисунок 1. *Пример выполнения процедуры ограниченного перебора*

Глубина перебора ограничивается заданным числом *n*, которое указывает максимальное количество физических элементов, среди которых можно производить переназначение (то есть при *n=2* назначенные элементы могут сниматься и переназначаться одновременно не более чем с двух физических серверов). На рис. 1 слева изображена ситуация, в которой виртуальная машина, требующая 4 ядра, не может быть размещена ни на один из физических серверов, несмотря на то, что суммарного количества ядер достаточно для ее размещения. Цветом закрашены размещенные на сервера R1, R2, R3 виртуальные машины и для каждой машины указано количество занимаемых ядер, белый прямоугольник - количество свободных ядер на сервере. При переназначении виртуальных машин размещенных на сервера R1, R2 фрагментация устраняется, и виртуальная машина может быть размещена. То есть в этом случае достаточно глубины перебора *n=2*. На рис. 1 справа изображена ситуация, в которой виртуальная

машина требует 5 ядер. В этом случае если глубина перебора равна 2 машина не может быть размещена. Но если задана глубина перебора *n=3*, то виртуальная машина будет размещена. То есть, увеличивая допустимую глубину перебора можно повышать точность алгоритма, но при этом увеличивается его вычислительная сложность.

В таблице 1 приведены результаты работы алгоритма для двух случаев:

1. Миграция работающих виртуальных ресурсов запрещена.
2. Нет ограничений на время миграции виртуальных ресурсов, поменявших свое расположение. В этом случае процедура построения плана миграции не используется.

В таблице приведены процент размещенных запросов, учитывая и ранее размещенные, и новые запросы; загрузка вычислительных ресурсов.

	Нет ограничений на время миграции			Миграция запрещена		
	Загрузка CPU	Загрузка RAM	Процент размещенных запросов	Загрузка CPU	Загрузка RAM	Процент размещенных запросов
Тест1	0.97	0.33	100%	0.83	0.29	75%
Тест2	0.97	0.33	100%	0.83	0.29	75%
Тест3	0.89	0.49	85%	0.83	0.47	75%
Тест4	0.86	0.48	79%	0.83	0.47	75%
Тест5	0.85	0.47	77.5%	0.83	0.47	75%

Таблица 1. *Эффективность использования ресурсов ЦОД*

При наличии миграции процент используемых ресурсов ЦОД на всех тестах выше, чем при отсутствии миграции. Выигрыш будет тем больше, чем чаще вызывается процедура ограниченного перебора.

Но если есть ограничение на время миграции, то результаты работы алгоритма будут тем хуже, чем меньше время, отведенное для миграции. Данная зависимость вытекает из результатов работы алгоритма, представленных в таблице 2.

	Тест1	Тест2	Тест3	Тест4	Тест5
Время миграции = 500 усл. ед.					
Загрузка CPU	0.97	0.97	0.89	0.86	0.85
Загрузка RAM	0.33	0.33	0.49	0.48	0.47
Процент размещенных запросов	100%	100%	85%	79%	77.5%
Время миграции = 400 усл. ед.					
Загрузка CPU	0.94	0.95	0.89	0.86	0.85
Загрузка RAM	0.325	0.328	0.49	0.48	0.47
Процент размещенных запросов	95%	97%	85%	79%	77.5%
Время миграции = 300 усл. ед.					
Загрузка CPU	0.9	0.91	0.89	0.86	0.85
Загрузка RAM	0.31	0.32	0.49	0.48	0.47
Процент размещенных запросов	87.5%	90%	85%	79%	77.5%
Время миграции = 200 усл. ед.					
Загрузка CPU	0.87	0.87	0.89	0.86	0.85
Загрузка RAM	0.3	0.3	0.49	0.48	0.47
Процент размещенных запросов	82.5%	82%	85%	79%	77.5%
Время миграции = 100 усл. ед.					
Загрузка CPU	0.85	0.85	0.86	0.86	0.85
Загрузка RAM	0.3	0.29	0.48	0.48	0.47
Процент размещенных запросов	77.5%	78%	80%	79%	77.5%
Время миграции = 50 усл. ед.					
Загрузка CPU	0.85	0.84	0.84	0.85	0.85
Загрузка RAM	0.3	0.29	0.47	0.47	0.47
Процент размещенных запросов	77.5%	76%	76.2%	77.5%	77.5%

Таблица 2. *Эффективность использования ресурсов ЦОД при наличии ограничения на время миграции*

Для того чтобы показать влияние миграции на качество получаемого алгоритмом планирования решения, были выбраны такие данные, на которых бы часто вызывалась процедура ограниченного перебора. Физические ресурсы представляют собой кластеры из 3 серверов и коммутатора. На эти сервера назначены виртуальные машины таким образом, что без вызова процедуры ограниченного перебора назначение новой виртуальной машины невозможно. Описанные кластеры связаны в кольцо каналами, которые проходят между коммутаторами.

В таблице 3 представлено описание параметров для генерации исходных данных: количество серверов в топологии; интервалы, в которых могут меняться CPU и RAM у виртуальных машин и у физических серверов; пропускная способность физических каналов связи (измеряется в Мб/с); доля размещенных в ЦОД запросов на момент запуска алгоритма распределения ресурсов в ЦОД (учитываются и старые, и новые запросы).

	Физические сервера				Виртуальные машины		Доля размещ. запросов
Тест, №	Кол-во	CPU	RAM	Каналы	CPU	RAM	0.75
1	30	10	8192	50	1-5	1024	0.75
2	45	10	8192	80	1-5	1024	0.75
3	60	10	5120	150	1-5	1024	0.75
4	90	10	5120	200	1-5	1024	0.75
5	120	10	5120	300	1-5	1024	0.75

Таблица 3. *Описание параметров для генерации исходных данных*

На рис. 2 представлена зависимость доли размещенных запросов от ограничения на время миграции. В последних тестах процедура ограниченного перебора не находила решения, поэтому результаты при разном ограничении на время миграции примерно одинаковые.

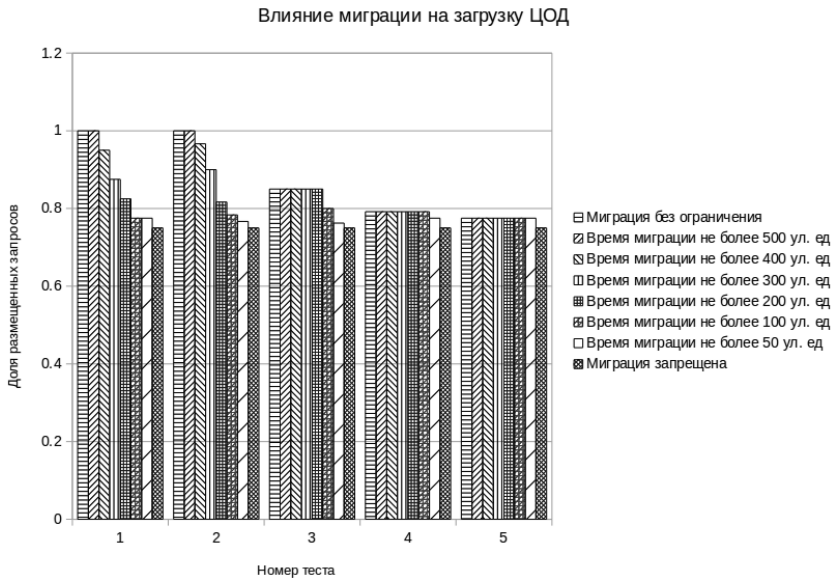

Рисунок 2. *Доля размещенных запросов при разном ограничении на времена миграции: 50, 100, 200, 300, 400, 500 условных единиц*

В данной статье было исследовано влияние миграции на эффективность работы ЦОД по критериям загрузка физических ресурсов и количество размещенных запросов. Если миграция допустима, то повышается загрузка ресурсов и увеличивается количество размещенных запросов. Однако миграция виртуальных ресурсов без их останова может значительно увеличивать нагрузку на сетевые ресурсы ЦОД. Чтобы не нарушить запрошенные критерии качества обслуживания других запросов может потребоваться значительное время для миграции перемещаемых виртуальных ресурсов. Это может приводить к тому, что время размещения новых виртуальных ресурсов будет неприемлемо большим. Поэтому время миграции надо обязательно ограничивать, несмотря на то, что это может приводить к уменьшению загрузки физических ресурсов и количеству размещенных запросов. Другой альтернативой является миграция виртуальных ресурсов с их остановом.

Литература

1. Amies A, Sluiman H., Tong Q.G., et al. Developing and Hosting Applications on the Cloud. Boston: IBM Press, 2012.
2. Wustenhoff E. Service level agreement in the data center. Sun BluePrints. Sun Microsystems Professional Series. 2002.
3. Зотов И. А., Костенко В. А. Алгоритм распределения ресурсов в центрах обработки данных с единым планировщиком для различных типов ресурсов // Известия РАН. Теория и системы управления, 2015., № 1, С. 61-71.
4. В. А. Костенко. Алгоритмы комбинаторной оптимизации, сочетающие жадные стратегии и ограниченный перебор// Известия РАН. Теория и системы управления, 2017, № 2, с. 48–56.

Костенко В.А., Шостик А.В.

Алгоритмы имитации отжига для задачи отображения виртуальных ресурсов на физические в центрах обработки данных

Введение

В работе рассматриваются центры обработки данных (ЦОД), работающие в режиме Infrastructure-as-a-Service (IaaS) [1]. При работе ЦОД в режиме IaaS необходима возможность задания критериев качества сервиса (SLA) [2] для всех типов ресурсов: систем хранения данных, вычислительных и сетевых ресурсов. Вычислительные ресурсы, системы хранения данных и сетевые ресурсы должны рассматриваться, как планируемые типы ресурсов. В ходе эксплуатации ЦОД происходит назначение поступающих пользовательских запросов на предоставление ресурсов (виртуальных ресурсов) на физические ресурсы. Для получаемых отображений виртуальных ресурсов на физические ресурсы ЦОД необходимо гарантированное выполнение запрошенных SLA.

В статье рассматривается применимость алгоритмов имитации отжига для решения задачи распределения виртуальных ресурсов между физическими ресурсами в ЦОД.

В разделе 1 приведена математическая постановка задачи отображения виртуальных ресурсов на физические. Раздел 2 содержит описание алгоритма распределения ресурсов. В разделе 3 приводятся результаты экспериментального исследования свойств предложенного алгоритма имитации отжига и сравнение результатов его работы с жадным алгоритмом распределения ресурсов.

1. Постановка задачи

Для формулировки задачи отображения виртуальных ресурсов на физические ресурсы в ЦОД будем использовать математическую модель функционирования ЦОД, используемую в работах [3-5]. Ниже приведем из [3-5] основные понятия, которые необходимы для формулировки задачи и описания предлагаемого алгоритма имитации отжига.

Модель физических ресурсов ЦОД задается размеченным графом $H = (P \cup M \cup K, L)$, где P – множество вычислительных узлов, M – множество хранилищ данных, K – множество коммутационных элементов сети обмена ЦОД, L – множество физических каналов

передачи данных. На множествах P, M, K и L определены векторные функции скалярного аргумента, задающие соответственно характеристики вычислительных узлов, хранилищ данных, коммутационных элементов и каналов передачи данных.

Ресурсный запрос задается размещенным графом $G = (W \cup S, E)$, где W – множество виртуальных машин, S – множество виртуальных систем хранения данных (storage-элементов), E – множество виртуальных каналов передачи данных. На множествах *W, S* и *E* определены векторные функции скалярного аргумента, задающие характеристики запрашиваемого виртуального элемента (требуемое качество сервиса (SLA)).

Назначением ресурсного запроса будем называть отображение:

$A : G \to H = \{W \to P, S \to M, E \to \{K, L\}\}$. .

Между характеристиками запросов и соответствующих характеристик физических ресурсов возможно три типа отношений. Обозначим через *x* характеристику элемента запроса и через *y* соответствующую ей характеристику физического ресурса. Тогда эти отношения можно записать следующим образом:

1. Недопустимость перегрузки «емкости» физического ресурса:

$$\sum_{i \in R_j} x_i \le y_j,$$

здесь R_j - множество запросов, назначенных на выполнение на физическом ресурсе *j*.

2. Соответствие типа физического и виртуального ресурса:

$$x_i = y_j.$$

3. Наличие требуемых характеристик у физического ресурса:

$$x_i \le y_j.$$

Отображение A называется корректным, если для всех физических ресурсов и всех их характеристик выполняются отношения 1-3.

Остаточным графом доступных ресурсов H_{res} называется граф H, в котором переопределены значения функций по характеристикам, которые должны удовлетворять отношению 1. Значения каждой характеристики физического ресурса уменьшается на сумму значений соответствующей характеристики виртуальных ресурсов, назначенных на этот физический ресурс.

Входными данными задачи построения отображения виртуальных ресурсов на физические ресурсы являются:

1) Остаточный граф доступных ресурсов $H_{res} = (P \cup M \cup K, L)$;

2) Множество назначенных запросов $Ten = \{(T_i, flag_i)\}$, где $flag_i = (0 \mid 1)$ – возможность миграции запроса T_i;

3) Отображение назначенных запросов: $Assigned = \{A : T_i \to H_{res}\}$;

4) Множество запросов, поступивших планировщику $Z = \{G_i\}$;

5) Множество дополнительных виртуальных ресурсов (элементов запроса) к назначенным запросам $V = \{(W, i_w), (S, i_s), (E, i_e)\}$, где i_w, i_s, i_e – индексы запросов, к которым относятся данные виртуальные ресурсы.

Введем множество $R = Z \cup V$ – все поступившие запросы для назначения на физические ресурсы ЦОД. Требуется из множества R разместить на выполнение в ЦОД максимальное количество запросов.

2. Алгоритм назначения запросов на физические ресурсы ЦОД

Алгоритм имитации отжига применяется для назначения виртуальных машин и виртуальных хранилищ данных. Общая схема алгоритма для решения задачи отображения виртуальных ресурсов на физические выглядит следующим образом:

1. Назначение виртуальных машин на вычислительные узлы ЦОД и storage-элементов на хранилища данных ЦОД.
2. Для полученных отображений виртуальных машин и storage-элементов на физические ресурсы построение отображения виртуальных каналов запросов на физические каналы передачи данных и коммутационные элементы.

2.1 Назначение виртуальных машин и storage-элементов

В литературе задачи назначения виртуальных машин и storage-элементов на физические ресурсы известны, как задачи упаковки предметов в контейнеры [6]. Предлагаемый алгоритм решения данной подзадачи представляет собой алгоритм имитации отжига.

Общая схема алгоритма имитации отжига следующая [7].

Шаг 1. Задать начальное корректное решение X^0 и считать его текущим вариантом решения ($X = X^0$).

Шаг 2. Установить начальную температуру T_0 , приняв ее текущей ($T = T_0$).

Шаг 3. Применить операции преобразования решения к текущему решению X и получить новый корректный вариант решения X', если это решение является лучшим из ранее найденных решений, то запомнить его.

Шаг 4. Найти изменение функции оценки качества решения:
$\Delta F = F(X') - F(X)$.

- если $\Delta F \le 0$ (решение улучшилось), то новый вариант решения считать текущим ($X = X'$);
- если $\Delta F > 0$ (решение ухудшилось), то принять с вероятностью $p = e^{-\frac{\Delta F}{T}}$ в качестве текущего решения новый вариант решения X'.

Шаг 5. Повторить заданное число раз шаги 3 и 4 без изменения текущей температуры.
Шаг 6. Если критерий останова выполнен, завершить работу алгоритма.
Шаг 7. Понизить текущую температуру в соответствии с выбранным законом понижения и перейти к шагу 3.

Для построения алгоритма имитации отжига для назначения виртуальных машин и storage-элементов необходимо определить следующие операции:

1. Генерация начального решения $X = X^0$.
2. Операция мутации (изменение текущего отображения виртуальных ресурсов на физические) $M : X \to X'$.
3. Функция оценки текущего отображения (целевая функция) $F(X)$.
4. Закон понижения температуры.

2.1.1 Генерация начального назначения виртуальных машин и storage-элементов

Рассматриваются три различных способа задания начального назначения запросов.

Начальное назначение запросов по стратегии *First-fit* – назначение первого подходящего запроса на первый подходящий ресурс.
Шаг 1. Из списка запросов *R* выбрать запрос, следующий по очереди. Если список пуст, завершить работу алгоритма.
Шаг 2. Из списка элементов запроса (виртуальные машины и виртуальные хранилища данных) выбрать следующий по списку элемент *vt*.

Шаг 2.1. В соответствии с типом элемента *vt* из списка узлов физических ресурсов выбрать случайный элемент *v* соответствующего типа.
Шаг 2.2. Если назначение *vt* на *v* возможно, назначить элемент *vt*, перейти к шагу 2. Если нет, вернуться к шагам 2.1, 2.2 *n* раз.
Шаг 2.3. Если назначение узла не завершилось успешно, удалить назначения всех элементов запроса, перейти к шагу 1.

Шаг 3. Перейти к шагу 1 – назначению следующего запроса.

Начальное назначение запросов по стратегии *Random-fit* – назначение случайного запроса на случайный ресурс.

Шаг 1. Из списка запросов *R* случайным образом выбрать запрос.

Шаг 2. Из списка элементов запроса (виртуальные машины и виртуальные хранилища данных) выбрать следующий по списку элемент *vt*.

Шаг 2.1. В соответствии с типом элемента *vt* из множества элементов физических ресурсов выбрать случайный элемент *v* соответствующего типа.

Шаг 2.2. Если назначение *vt* на *v* возможно, назначить элемент *vt*, перейти к шагу 2. Если нет, вернуться к шагам 2.1, 2.2 *n* раз.

Шаг 2.3. Если назначение элемента не завершилось успешно, удалить назначения всех элементов запроса, перейти к шагу 1.

Шаг 3. Перейти к шагу 1 – назначению следующего запроса.

Начальное назначение запросов на основе *жадных* стратегий.

Шаг 1. Упорядочить элементы запросов в порядке убывания их емкостей.

Шаг 2. Упорядочить элементы физических ресурсов в порядке увеличения их емкостей.

Шаг 3. Произвести назначение упорядоченных запросов на упорядоченные физические ресурсы согласно стратегии *First-fit*.

2.1.2 Операции перехода от одного отображения к другому (мутация)

Операция мутации включает в себя следующие процедуры:

1. Add. Назначение нового запроса.

   Общая схема данной процедуры:

   Шаг 1. Из списка неназначенных запросов случайным образом выбрать запрос *T*.

   Шаг 2. Из списка элементов запроса (виртуальные машины и виртуальные хранилища данных) выбрать следующий элемент *vt*.

   Шаг 2.1. В соответствии с типом элемента *vt* из множества элементов физических ресурсов выбрать случайный элемент *v* соответствующего типа.

   Шаг 2.2. Если назначение *vt* на *v* возможно, назначить узел *vt*, перейти к шагу 2. Если нет, вернуться к шагам 2.1, 2.2 *n* раз.

   Шаг 2.3. Если назначение элемента не завершилось успешно, удалить назначения всех элементов запроса.

2. Delete. Снятие назначения случайно выбранного запроса.

   Общая схема данной процедуры:

   Шаг 1. Из списка назначенных запросов случайным образом выбрать запрос *T*.

   Шаг 2. Из списка элементов запроса (виртуальные машины и виртуальные хранилища данных) выбрать следующий элемент *vt*.

Снять элемент *vt* с соответствующего физического элемента *v,* переопределить свободную емкость элемента *v.* Повторить шаг 2, пока список элементов запроса не пуст.

3. Move. Переназначение случайно выбранного запроса.

Производится удаление назначения случайно выбранного запроса согласно схеме операции Delete, затем назначение данного запроса на случайные ресурсы аналогично операции Add.

Операция мутации выполняет одну из трех перечисленных процедур, исходя из того, какие процедуры выполнялись ранее и с каким исходом (успешно или нет).

Целью операции мутации является назначение большего количества запросов, чем на предыдущих шагах. Схема, по которой операция мутации выбирает процедуру заключается в следующем: выполняется назначение новых запросов, пока возможно, затем перемещение некоторого случайного запроса на другой ресурс, попытка разместить новый запрос из списка неразмещенных запросов, и если перемещение не привело к успеху, возврат предыдущего назначения случайного запроса.

2.1.3 Целевая функция

Целевая функция, то есть критерий качества отображения для задачи построения назначения запросов на физические ресурсы – количество размещенных запросов в сети физических ресурсов ЦОД. В предложенном алгоритме максимизируется.

2.1.4 Закон понижения температуры

В качестве закона понижения температуры было проведено сравнительное исследование следующих функций [8]:

$$t_i = \frac{T_0 \ln(1+i)}{1+i},$$

$$t_i = \frac{T_0}{\ln(1+i)} - \text{закон Больцмана},$$

$$t_i = \frac{T_0}{1+i} - \text{закон Коши}.$$

2.2 Назначение виртуальных каналов на физические ресурсы сети

Назначения виртуальных каналов производятся с использованием жадных стратегий и стратегий ограниченного перебора согласно алгоритму, приведенному в работе [3].

Алгоритм строится на основе алгоритма Йена для задачи нахождения *k* кратчайших путей [9]. Запросы и виртуальные каналы в ходе построения маршрута выбираются согласно заданным жадным критериям: выбирается запрос с наибольшей суммарной пропускной

виртуальных каналов, после чего последовательно выбирается виртуальный канал с наибольшей пропускной способностью. В случае неуспеха назначения выбранного виртуального канала применяется процедура ограниченного перебора, которая заключается в переборе множеств ограниченной мощности из уже назначенных виртуальных каналов с целью их переназначения таким образом, чтобы данный виртуальный канал мог быть назначен. В случае неудачи назначения виртуального канала запрос снимается.

3. Результаты экспериментального исследования свойств алгоритма

Экспериментальное исследование было проведено с целью выявить зависимость качества работы алгоритма от выбора закона понижения температуры и сравнить работу алгоритма с результатами работы других алгоритмов распределения ресурсов в ЦОД, а именно алгоритмом случайного назначения запросов Random-fit, алгоритмом назначения первого подходящего запроса на первый подходящий ресурс First-fit и алгоритмом на основе жадных стратегий.

Критерии сравнения:

- число размещенных запросов,
- время работы алгоритмов.

Исследование было проведено в инструментальной среде, разработанной для моделирования распределения ресурсов в ЦОД.

3.1 Зависимость от выбора закона понижения температуры

Исследование зависимости качества работы алгоритма имитации отжига от выбора закона понижения температуры проводилось на тестах с возможностью полного назначения запросов, и в этом случае загрузка ресурсов составляла 100%. Физические ресурсы в этих тестах состояли из набора вычислительных узлов количеством $10n, n \in [1..8]$ с одинаковыми параметрами. Все запросы (всего их также $10n, n \in [1..8]$) в объединении представляли собой множество виртуальных машин. Каждый запрос имел суммарную производительность виртуальных машин, равную производительности вычислительного узла.

Результаты исследования зависимости количества размещенных запросов и времени работы алгоритма от выбора закона понижения температуры приведены на рисунках 1, 2. При использовании закона Больцмана алгоритм имитации отжига выполняет большое количество итераций и имеет большую вычислительную сложность. Однако это позволяет размещать больше запросов. Закон Больцмана можно использовать в случае, когда время работы алгоритма не критично.

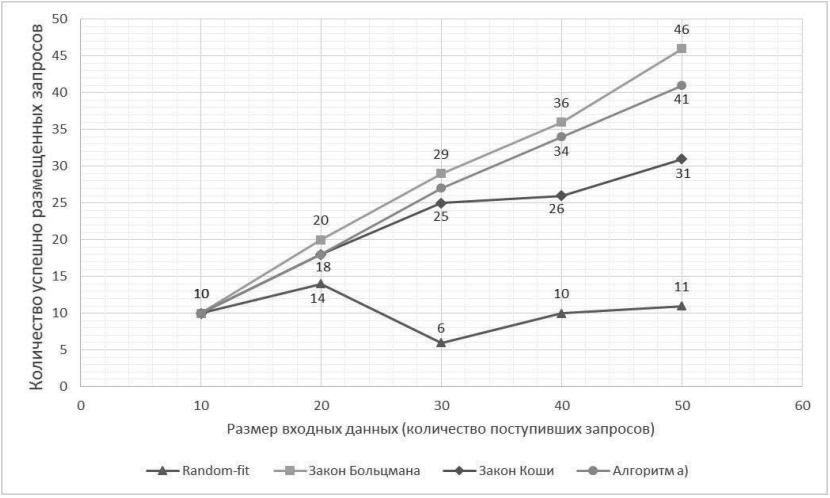

Рисунок 1. *Количество размещенных запросов в зависимости от выбора закона понижения температуры. Закон а):* $t_i = \frac{T_0 \ln(1+i)}{1+i}$

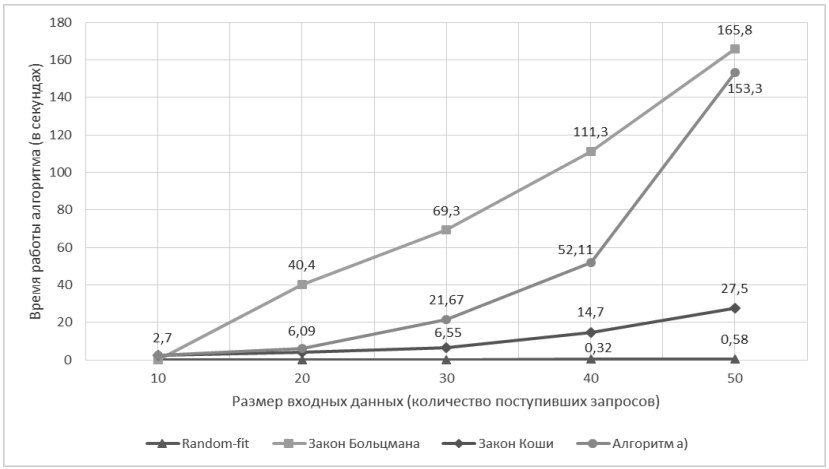

Рисунок 2. *Время работы алгоритма в зависимости от выбора закона понижения температуры. Закон а):* $t_i = \frac{T_0 \ln(1+i)}{1+i}$

3.2. Результаты экспериментального исследования алгоритма в сравнении с алгоритмами First-fit, Random-fit, жадным алгоритмом

Также было проведено исследование работы алгоритма в сравнении с результатами работы других алгоритмов распределения ресурсов в ЦОД: алгоритмом случайного назначения запросов *Random-fit*, алгоритмом назначения первого подходящего запроса на первый подходящий ресурс *First-fit* и алгоритмом на основе жадных стратегий.

Набор тестовых данных для такого исследования представлял собой совокупность произвольных физических ресурсов и запросов. Физические ресурсы ЦОД в исходных данных включают в себя вычислительные узлы, хранилища данных, сети обмена и имеют разные по величине параметры. Запросы состоят из достаточно большого количества элементов: виртуальные машины, storage-элементы, виртуальные каналы. Оптимальная загрузка в таком случае не является полной, так как суммарные емкости физических ресурсов одного типа больше, чем суммарные емкости всех запросов соответствующего им типа.

Результаты исследования (рис. 3) показали, что алгоритм имитации отжига улучшает качество отображений, получаемых жадным алгоритмом, однако имеет очень высокую вычислительную сложность, и поэтому его применение для задачи планирования вычислений в реальных ЦОД ограниченно допустимым временем работы. Оптимальный баланс между количеством размещаемых запросом и временем работы демонстрирует жадный алгоритм.

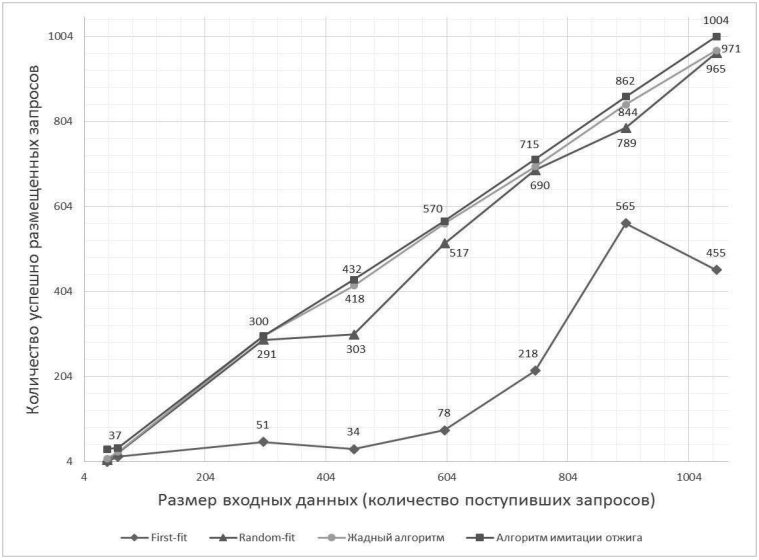

Рисунок 3. *Результаты работы алгоритма имитации отжига в сравнении с алгоритмами First-fit, Random-fit, жадным алгоритмом*

4. Заключение

В данной работе предложено решение задачи отображения виртуальных ресурсов на физические в ЦОД с использованием алгоритма имитации отжига. Результаты исследования свойств данного алгоритма показали высокую эффективность алгоритма имитации отжига по количеству размещаемых запросов, однако алгоритм требует больших временных затрат. Последнее существенно ограничивает его применение в реальных ЦОД.

Литература

1. Amies A, Sluiman H., Tong Q.G., et al. Developing and Hosting Applications on the Cloud. Boston: IBM Press, 2012.
2. Wustenhoff E. Service level agreement in the data center. Sun BluePrints. Sun Microsystems Professional Series. 2002.
3. Вдовин П.М., Костенко В.А. Алгоритм распределения ресурсов в центрах обработки данных с раздельными планировщиками для различных типов ресурсов // Известия РАН. Теория и системы управления. 2014. № 6. С. 80-93.
4. Зотов И. А., Костенко В. А. Алгоритм распределения ресурсов в центрах обработки данных с единым планировщиком для различных типов ресурсов // Известия РАН. Теория и системы управления. 2015. № 1. С. 61-71.
5. A. Plakunov, V. Kostenko. Data Center Resource Mapping Algorithm Based on the Ant Colony Optimization // Proceedings of the International Science and Technology Conference Modern Networking Technologies (MoNeTec), — Moscow, Russia, MAKS Pess, 2014. — pp.127–132.
6. Coffman E.G., Garey M.R., Johnson D.S. Approximation algorithms for bin packing: A survey // Approximation algorithms for NP-hard problems. Boston: PWS Publishing Co., 1996. P. 46-93.
7. Д.А.Зорин, В.А.Костенко. Алгоритм имитации отжига для решения задач построения многопроцессорных расписаний // Автоматика и телемеханика, 2014, № 10, С. 97-110.
8. Nourani Y., Andresen B. A comparison of simulated annealing cooling strategies //Journal of Physics A: Mathematical and General. – 1998. – Т. 31. – №. 41. – С. 8373.
9. Yen J. Y. Finding the k shortest loopless paths in a network // Management Science. 1971. T. 17. N. 11. P. 712-716.

Пинаева Н.М., Антоненко В.А.

Система управления версиями виртуальных сетевых функций в облачной платформе[1]

Введение

Виртуализация сетевых функций (Network Functions Virtualization, NFV) – концепция разделения сетевой функциональности и оборудования, которое ее реализует, с помощью технологии виртуализации физических ресурсов.

Данный подход отделяет программную реализацию сетевой функции от оборудования, на котором она выполняется. Достоинствами концепции являются возможности динамического масштабирования, автоматической конфигурации виртуальных сетевых функций, эффективного использования физических ресурсов и их перераспределения.

Стандартом архитектуры NFV платформ является высокоуровневая модель управления и оркестрации для виртуализации сетевых функций (NFV Managment And Orchestration, NFV MANO) [1], разработанная специальной группой Европейского Института Телекоммуникационных Стандартов – ETSI NFV Industry Specification Group. Она обеспечивает управление виртуальной инфраструктурой и планирование ресурсов, необходимых для создания сетевых сервисов и виртуальных сетевых функций. В описании высокоуровневой модели NFV MANO определяется три основных уровня архитектуры системы виртуализации сетевых функций, выстроенных в иерархию:

- виртуальная инфраструктура,
- виртуальная сетевая функция,
- сетевой сервис,

которые определены в [2] следующим образом.

- Виртуальная инфраструктура или инфраструктура виртуализации сетевых функций (Network Functions Virtualisation Infrastructure, NFVI) — совокупность аппаратного и программного обеспечения, образующего окружение, в котором могут быть установлены и работать виртуальные сетевые функции.

  Виртуальная инфраструктура обычно включает виртуальные вычислительные ресурсы, виртуальные хранилища данных, виртуальные сетевые ресурсы.

[1] Работа выполнена при финансовой поддержке РФФИ, грант № 16-07-01261.

- Виртуальная сетевая функция (Virtualised Network Function, VNF) — реализация сетевой функции, которая может быть установлена и работать на инфраструктуре NFVI.
  Под сетевой функцией понимают функциональный блок сетевой инфраструктуры, который обрабатывает сетевой трафик.
- Сетевой сервис (Network Service, NS) — композиция сетевых функций, определяемых своей функциональной и поведенческой спецификацией.

Для формального описания виртуальных сетевых функций используется TOSCA (Topology and Orchestration Specificationfor Cloud Applications) [3] — спецификация, которая позволяет описывать структуру облачных приложений и аспекты управления им. Описание функции с использованием данного стандарта называется TOSCA шаблоном. Данная спецификация позволяет описывать TOSCA шаблоны функций на языке YAML. Все необходимые данные для установки и функционирования VNF, включая TOSCA шаблон, объединяются в архив, называемый VNF пакетом. Таким образом, для предоставления новой VNF необходимо сформировать VNF пакет и загрузить его в облачную платформу. Данный VNF пакет в облачной платформе определяет VNF и будет использован при создании экземпляра VNF – непосредственном развертывании описанной функции на виртуальной инфраструктуре.

TOSCA оперирует понятием узла для описания различных частей приложений, основными типами узлов являются виртуальные машины и приложения, работающие на них. Под политикой в TOSCA понимаются правила обработки группы узлов, объединенные для достижения определенного результата. Примером политики является политика размещения, которая определяет, каким образом нужно разместить группы виртуальных машин (например, на одном вычислительном узле или, наоборот, обязательно на разных).

Наиболее интересной политикой в рамках данной статьи является политика обновления, которая необходима для управления обновлениями виртуальных сетевых функций. Политика обновления не имеет конкретного описания в TOSCA, так как она должна включать в себя описание некоторых параметров обновления, которые могут быть правильно проинтерпретированы облачной платформой, а также будут понятны и полезны пользователю.

Новой версией функции будем считать функцию с тем же именем, но отличным номером версии. Обновлением экземпляра виртуальной сетевой функции будем называть изменение существующего экземпляра функции таким образом, чтобы он точно соответствовал описанию из VNF пакета с новой версией.

Системой управления версиями виртуальных сетевых функций будем называть систему, обеспечивающую следующие возможности:

- поддержка нескольких версий одной виртуальной сетевой функции (включает поддержку базы данных с несколькими версиями одной виртуальной сетевой функции, возможность вывода всех зарегистрированных версий одной виртуальной сетевой функции, поддержку регистрации новой версии виртуальной сетевой функции и т.д.);
- выбор версии при создании экземпляра виртуальной сетевой функции;
- обновление экземпляров виртуальных сетевых функций в соответствии с некоторым описанием политики обновления до более новой версии.

NFV – это новая концепция, поэтому реализующие ее проекты, в основном, сейчас находятся в процессе разработки и внедрения. Таким образом, системы управления версиями виртуальных сетевых функций на данный момент слабо развиты. Так как почти все существующие приложения обовляются и часто эти обновления являются критическими (такие как обновления безопасности и исправление ошибок), то при виртуализации этих приложений система управления версиями буден необходима в облачной платформе.

В данной работе в качестве NFV платформы для разработки системы управления версиями VNF использована платформа C2. C2 (Cloud conductor) – облачная платформа, реализующая высокоуровневую модель ETSI NFV MANO, которая поддерживает полный жизненный цикл виртуальных сетевых сервисов.

К разрабатываемой системе управления версиями предъявляются следующие требования:

- возможность описания сервиса с использованием TOSCA,
- возможность задания последовательности обновления виртуальных машин облачного сервиса,
- продолжение работы сервиса во время обновления,
- возможность отката к предыдущей версии (как при неудачном обновлении, так и по инициативе пользователя до завершения обновления),
- применение любых изменений функции при обновлении (в одной из существующих систем невозможно, например, уменьшение количества ядер виртуальной машины при обновлении).

В данной статье рассматривается разработка и реализация соответствующей указанным требованиям системы управления версиями виртуальных сетевых функций в облачной платформе, интеграция полученной системы в облачную платформу C2.

2. Обзор существующих решений

На данный момент существует всего две системы управления версиями: в системах Microsoft Azure и Cloudify. Целью данного обзора является выявление достоинств и недостатков существующих решений для использования полученных результатов при разработке системы управления версиями VNF. Критерии обзора – проверка соответствия систем управления версиями указанным во введении требованиям к разрабатываемой системе.

2.1 Система обновления облачных сервисов Microsoft Azure

Облачные сервисы Microsoft Azure [4] предназначены для поддержки масштабируемых облачных сервисов. Облачные сервисы размещаются на виртуальных машинах, на которых пользователь может установить собственное программное обеспечение, а также удаленно управлять ими.

Обновление облачного сервиса включает три шага. Первый – загрузка на виртуальные машины новых бинарных и конфигурационных файлов и новой версии операционной системы. Далее, Microsoft Azure резервирует вычислительные и сетевые ресурсы для построения новой топологии. Последним шагом Microsoft Azure выполняет последовательное обновление.

Последовательное обновление протекает следующим образом. Microsoft Azure объединяет виртуальные машины в логические группы, называемые доменами обновления. Домен обновления – логическое множество виртуальных машин, которые обновляются как группа. Microsoft Azure обновляет облачный сервис по одному домену обновления, что позволяет машинам из других доменов продолжать обрабатывать трафик. Внутри одного домена обновления производится обновление всех виртуальных машин следующим образом: виртуальная машина останавливается, обновляется и становится снова доступной. Пользователь может задавать последовательность обновления виртуальных машин с помощью задания доменов обновлений.

Microsoft Azure обеспечивает гибкость в управлении сервисами во время обновления, предлагая возможность отката к предыдущей версии. Откат может быть произведен только в процессе обновления, пока хотя бы одна виртуальная машина не обновлена.

Существует несколько ситуаций, в которых откат невозможен:

- если в процессе обновления был увеличен объем ресурсов для какой-либо виртуальной машины;
- если обновление является вертикальным масштабированием с уменьшением объема ресурсов;
- если обновление успешно завершилось.

Данная система не поддерживает следующие изменения во время обновления:

- изменение количества виртуальных машин, принадлежащих одному домену обновления,
- уменьшение количества ресурсов (количество ядер, оперативной памяти, размер дискового пространства) для виртуальных машин.

2.2 Система обновления облачных сервисов Cloudify

Cloudify [5] – платформа для оркестрации облачных приложений. Она предоставляет возможность автоматизации управления жизненным циклом приложений и сервисов, мониторинга основных параметров размещенного приложения, обнаружения проблем и отказов и др. Приложения в Cloudify формально описывают с использованием TOSCA и состоят из множества узлов и связей между ними, где основными типами узлов являются виртуальные машины и сети. Cloudify предоставляет возможность обновления размещенного приложения.

Узлы и связи, присутствовавшие в предыдущей версии приложения, но отсутствующие в новой версии считаются неактуальными. Стандартная последовательность действий при обновлении следующая:

- удаление всех неактуальных связей,
- удаление всех неактуальных узлов,
- установка и конфигурация новых узлов,
- создание новых связей.

Основные единицы обновления – узлы и связи. Узлы, как и все их свойства, могут быть добавлены или удалены. Связи могут быть удалены или добавлены в зависимости от добавляемых и удаляемых узлов, но также они могут быть добавлены или удалены независимо. Дополнительно есть возможность изменить порядок связей для узла.

Если стандартный порядок проведения обновления не подходит пользователю, Cloudify позволяет создать собственный алгоритм для проведения обновления. Такой алгоритм должен быть оформлен специальным образом как функция на языке Python, которая импортирует модуль, позволяющий взаимодействовать с некоторыми сервисами Cloudify. Таким образом, пользователю предоставляется возможность контролировать порядок обновления узлов.

В случае если процесс обновления завершился с ошибкой, есть возможность откатиться к предыдущей версии сервиса, но при любых других условиях нет возможности инициировать откат.

Если новая версия сервиса содержит неподдерживаемые изменения, обновление невозможно, и пользователь получит уведомление об этом. Список неподдерживаемых изменений следующий:

- изменение типа узла,
- изменение свойств связей,
- перенос программного обеспечения с одной виртуальной машины на другую,
- никакие изменения групп, политик и триггеров на данный момент не поддерживаются.

Соответствие рассмотренных систем указанным критериям представлено в таблице 1. Из таблицы видно, что систем, удовлетворяющих указанным требованиям, на данный момент не представлено.

Критерий	Microsoft Azure	Cloudify
Описание сервиса с использованием TOSCA	–	+
Задание последовательности обновления виртуальных машин	+	+
Продолжение работы сервиса во время обновления	+	+
Возможность отката к предыдущей версии	+/–	+/–
Применение любых изменений функции при обновлении	–	–

Таблица 1. *Результаты обзора*

В таблице 1 +/– означает возможность отката только при определенных условиях.

3. Разработанная система контроля версий

В данном разделе будет описана разработанная система управления версиями VNF.

C2 (Cloud conductor) – облачная платформа, реализующая высокоуровневую модель ETSI NFV MANO, которая поддерживает полный жизненный цикл виртуальных сетевых сервисов.

Управление обновлениями VNF является опциональной функцией модуля платформы C2, который называется VNF менеджер (VNFM), поэтому разрабатываемая система управления версиями является его частью.

Основным назначением VNF менеджера является управление жизненным циклом функций. Для обеспечения возможности управления жизненным циклом VNF необходимо уметь описывать саму VNF и действия, которые нужно предпринять на каждом этапе ее жизненного цикла. Для описания сетевых функций используется спецификация TOSCA на языке YAML, а само описание VNF с ее использованием называется TOSCA шаблоном.

В рамках работы было разработано описание (синтаксис) политики обновления, которое позволяет:

- производить любые изменения функции при обновлении;
- максимально уменьшать время недоступности сервиса и обеспечивать продолжение работы во время обновления;
- проводить срочные обновления (обновление без диалога с пользователем);
- проводить откат к предыдущей версии во время обновления;
- задавать последовательность обновления узлов сервиса;
- запрашивать входные параметры при обновлении.

Разработанная система управления версиями является частью модуля VNF Manager, также разработанного авторами данной работы [6]. В рамках данной работы были добавлены два класса: обработчик сообщения-запроса на обновление экземпляра функции (UpdateVNFIMessage) и вспомогательный класс для обновления TOSCA узла (UpdateNode). Также была изменена структура базы данных, и обработчики некоторых сообщений для возможности задания параметра версии функции.

Полученная система имеет возможность описания сервиса с использованием TOSCA, как часть VNFM. Остальные требования удовлетворяются благодаря возможностям разработанной политики обновления.

4. Экспериментальное исследование

Характеристиками обновления назовем следующие параметры:

- время недоступности сервиса при проведении данного обновления,
- время проведения обновления,
- количество дополнительно задействованных виртуальных ресурсов при обновлении:
  - количество ядер процессора,
  - объем оперативной памяти,
  - объем виртуального диска,
- время восстановления при неудачном обновлении.

Целью данного исследования является измерение характеристик нескольких различных обновлений одной виртуальной сетевой функции для демонстрации зависимости между ними.

В рамках данного исследования измерим значения характеристик для нескольких различных обновлений. В качестве основной версии функции возьмем функцию suricata 1.0. Функция создана на базе приложения suricata [7], являющегося системой обнаружения и предотвращения вторжений. Для визуализации результатов работы suricata используется веб-интерфейс snorby [8].

Описание функции suricata содержит следующую основную информацию:

- топология функции, представленная на рисунке 1,
- заданы требования к виртуальным машинам snorby_server и suricata_server,
- заданы два узла-приложения: snorby и suricata, которые должны быть запущены на виртуальных машинах snorby_server и suricata_server соответственно.

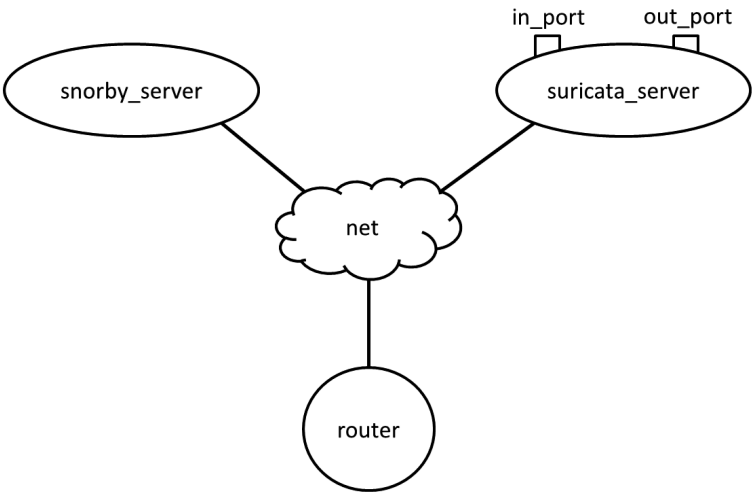

Рисунок 1. *Топология функции suricata.*

Для функции suricata 1.0 были описаны три новых версии. Обозначим номера этих версий как 2.a, 2.b и 2.c. Обновления до любой из версий будет производиться с версии 1.0. Для каждой из новых версий были созданы по три политики обновления: без дополнительных параметров (или простое обновление), с возможностью отката и с дублированием.

Для всех описанных политик было проведено обновление с версии 1.0. Для измерения времени восстановления при неудачном обновлении были инициированы ошибки обновления. На представленных ниже рисунках по горизонтальной оси расположены версии функции suricata.

На рисунке 2 представлены результаты измерений времени недоступности сервиса при различных обновлениях. В рамках исследования также были измерены еще две величины: время обновления ПО и время пересоздания функции. Время обновления ПО – время, которое занимает выполнение скриптов обновления, без учета их копирования на машину и других накладных расходов. Это время является аналогом времени недоступности сервиса при его работе вне облачной платформы. При отсутствии системы управления версиями, единственная возможность обновить экземпляр функции – удалить старый экземпляр и создать новый. Время этого процесса будет являться аналогом времени недоступности сервиса при отсутствии системы управления версиями.

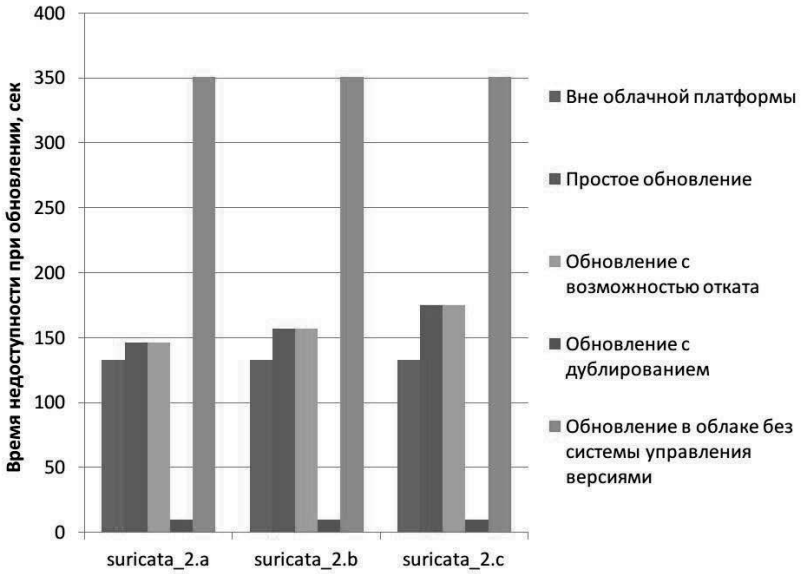

Рисунок 2. *Время недоступности сервиса при обновлении*

Таким образом, на рисунке 2 видна разница времени недоступности сервиса при обновлении вне облачной платформы, при проведенном в рамках исследования обновлении и в облачной платформе без системы управления версиями. Минимальное время недоступности достигается при обновлении с дублированием.

В клетках таблицы 2 приведено значение времени восстановления при неудачном обновлении экземпляра функции с номером из верхней строки и политикой обновления из первого столбца. Для первой политики восстановление функционирования экземпляра функции невозможно, поэтому время восстановления в таблице обозначено символом «∞».

Время восстановления, сек	2.a	2.b	2.c
Простое обновление	∞	∞	∞
Обновление с возможностью отката	126	135	135
Обновление с дублированием	0	0	0

Таблица 2. *Время восстановления при неудачном обновлении*

Заметим, что минимальное время недоступности (обновление с дублированием) влечет и минимальное время восстановления, а простое обновление не предоставляет возможности восстановления экземпляра функции.

На рисунке 3 приведено время обновления виртуальных сетевых функций, которое характеризует нагрузку на систему управления версиями.

Можно заметить, что при уменьшении времени восстановления, время обновления, наоборот, увеличивается, что обусловлено необходимостью запоминать состояние функции для возможности проведения отката и копировать экземпляр функции при дублировании.

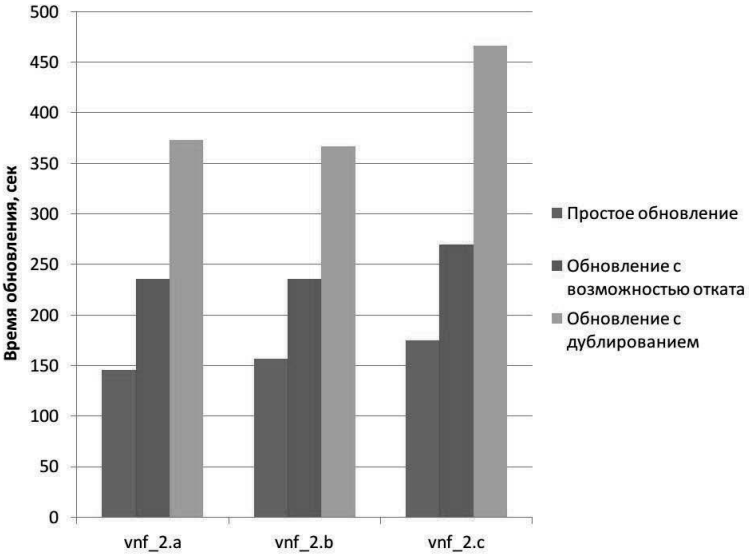

Рисунок 3. *Время обновления виртуальной сетевой функции*

Количество дополнительно задействованных виртуальных ресурсов совпадает для всех трех новых версий функции и зависит в данном случае только от номера политики обновления. В таблице 3 показаны результаты измерений, где количество виртуальных ядер обозначено как «CPU», объем оперативной памяти – «RAM», а объем жесткого диска – «Disk». Из таблицы 3 видно, что простое обновление не требует никаких дополнительных ресурсов, обновление с возможностью отката дополнительно требует некоторый объем диска, необходимый для хранения состояния виртуальных машин до обновления. Обновление с дублированием также требует запоминания состояния виртуальных машин для создания их копий, а также количество дополнительных ресурсов, равное количеству ресурсов, необходимых для работы экземпляра функции предыдущей версии.

	CPU, шт.	RAM, Гб	Disk, Гб
Простое обновление	0	0	0
Обновление с возможностью отката	0	0	5,1
Обновление с дублированием	2	2	65,1

Таблица 3. *Количество дополнительных ресурсов при обновлении*

По результатам проведенного исследования были показаны следующие основные зависимости характеристик обновления:

- время обновления при наличии системы управления версиями меньше времени обновления с отсутствием такой системы при любом из представленных обновлений;
- минимальное время недоступности сервиса влечет минимальное время восстановления и соответствует политике обновления с дублированием;
- время обновления минимально для политики без дополнительных параметров, увеличивается при переходе к обновлению с возможностью отката и увеличивается еще больше при обновлении с дублированием; имеет обратную зависимость со временем восстановления;
- количество дополнительно задействованных виртуальных ресурсов равно нулю при обновлении без дополнительных параметров; обновление с возможностью отката требует дополнительный объем дискового пространства; обновление с дублированием требует такое же количество ресурсов, как обновляемый экземпляр функции, увеличенный на объем диска, как при обновлении с возможностью отката.

Заключение

В рамках данной работы было разработано описание политики обновления на TOSCA, разработана и реализована система управления версиями, интерпретирующая политики, соответствующие полученному описанию. Данная система была интегрирована в облачную платформу С2.

Было проведено исследование, демонстрирующее зависимость характеристик обновления для нескольких различных обновлений виртуальной сетевой функции. Данное исследование показало преимущества использования разработанной системы: она позволяет уменьшить время недоступности сервиса при проведении обновления, позволяет изменять время обновления, количество дополнительно задействованных виртуальных ресурсов и время восстановления при неудачном обновлении с помощью различных описаний политики обновления.

Разработанная система также позволяет производить откат к предыдущей версии в любой момент до окончания обновления и применять любые изменения в новой версии виртуальной сетевой функции.

Литература

1. Quittek J. et al. Network Functions Virtualisation (NFV)-Management and Orchestration //ETSI, GROUP SPECIFICATION GS NFV-MAN 001 V1. 1.1. – 2014.
2. Han B., Gopalakrishnan V., Ji L. Network Functions Virtualisation (NFV); Terminoogy for Main Concepts in NFV //ETSI GS NFV. – 2013.
3. ISG N. F. V. Network Functions Virtualisation (NFV)-Virtual Network Functions Architecture. – ETSI, Tech. Rep, 2013.
4. Azure compute options – Cloud Services | Microsoft Docs [PDF] (https://opbuildstorageprod.blob.core.windows.net/output-pdf-files/en-us/Azure.azure-documents/live/cloud-services.pdf)
5. Updating a Deployment [HTML] (http://docs.getcloudify.org/3.4.0/manager/update-deployment/)
6. Пинаева Н. М., Антоненко В. А. Разработка и реализация системы управления виртуальными сетевыми функциями в облачной платформе // Программные системы и инструменты. Тематический сборник / Под ред. Р.Л. Смелянский. — Т. 16 из Программные системы и инструменты. — МАКС Пресс Москва, 2016. — С. 64–71.
7. Kuswanto D. Analisys Design Intrusion Prevention System (IPS) Based Suricata //PROCEEDING IC-ITECHS 2014. – 2015. – Т. 1. – №. 01.
8. Heenan R., Moradpoor N. Introduction to Security Onion. – 2016.

Антипина Е.А., Балашов В.В.

Обобщенная задача о мультипликативном рюкзаке со связями между объектами[1]

Введение

Задача о рюкзаке представляет из себя NP-трудную задачу комбинаторной оптимизации. Базовый смысл данной задачи заключен в упаковке максимального по ценности набора объектов в рюкзак с ограниченной емкостью. В работе [2] представлены большое число существующих типов задач о рюкзаке и их решения.

Наиболее расширенной является обобщенная задача о мультипликативном рюкзаке (OMP), в которой требуется упаковать максимальный по ценности набор объектов в предоставленный набор рюкзаков, каждый из которых имеет свою емкость. Ценность и объем объектов в таких задачах в каждом из рюкзаков различны (зависят от рюкзака, в который упакован объект). В [2] приведен ряд алгоритмов, решающих задачу OMP, в том числе алгоритм, основанный на методе ветвей и границ, строящий точное решение задачи. Этот алгоритм имеет экспоненциальную сложность относительно числа рюкзаков и объектов. Тем не менее имеет смысл оптимизация алгоритма с целью увеличения размерности задач, которые могут быть решены за приемлемое время с помощью данного алгоритма.

В данной статье представлена модификация обобщенной задачи о мультипликативном рюкзаке, а также предложен подход к решению модифицированной задачи на основе метода ветвей и границ. Основное отличие модифицированной задачи заключается в наличии связей между объектами, представленными для распределения по рюкзакам. То есть ценность объекта зависит не от того, в какой рюкзак он упакован, а от того, с какими другими объектами он упакован вместе. Для каждой пары объектов задана неотрицательная числовая характеристика — связь между ними. Необходимо упаковать объекты, максимизируя суммарную связь внутри рюкзаков. Задача такого вида возникает, в частности, при распределении вычислительной нагрузки в модульных вычислительных системах [6] в рамках построения расписания вычислений для таких ВС.

Структура данной статьи следующая. Раздел 1 содержит формальную постановку задачи OMP со связями между объектами. В

[1] Работа выполнена при финансовой поддержке РФФИ, грант № 17-07-01566.

разделе 2 представлено соответствие задачи распределения вычислительной нагрузки в модульных ВС задаче ОМР со связями. Раздел 3 содержит описание схем подсчета оценок для метода ветвей и границ. В разделе 4 представлены схемы доказательств корректности описанных оценок. Раздел 5 содержит оценку сложности предложенного алгоритма. В разделе 6 приведены результаты экспериментального исследования алгоритма на двух типах входных данных.

1. Формальная постановка обобщенной задачи о мультипликативном рюкзаке со связями между объектами

Входными данными для обобщенной задачи о мультипликативном рюкзаке со связями являются:

- множество рюкзаков K: K=$\{ K_1, K_2, ..., K_{N_K} \}$, |K|= N_K ;
- для каждого рюкзака задана его ёмкость: $C_n > 0,\ \forall n \in [1; N_K]$;
- множество объектов M: M=$\{ M_1, M_2, ..., M_{N_M} \}$, |M|= N_M ;
- вклад w_{in} каждого объекта в заполнение каждого из рюкзаков (объем объекта зависит от рюкзака, в который он упакован): $w_{in} \geq 0, \forall i \in [1; N_M], \forall n \in [1; N_K]$;
- числовая характеристика связи l_{ij} между каждой парой объектов из множества M: $l_{ij} \geq 0, \forall i, j \in [1; N_M]$, (если $l_{ij} = 0$, значит между объектами M_i и M_j связи нет).

Необходимо упаковать все объекты множества M в рюкзаки множества K, максимизируя суммарную связь между объектами внутри рюкзаков, соблюдая ограничения на ёмкость рюкзака:

- $S = \max \left(\sum_{n=1}^{N_T} \sum_{i=1}^{N_M - 1} \sum_{j=i+1}^{N_M} l_{ij} * X_{in} * X_{jn} \right)$,

$$X_{in} = \begin{cases} 1, \text{если } \mathrm{M_i} \in K_n \\ 0, \text{иначе} \end{cases}, \quad \forall i \in [1; N_M];$$

(x_{in} — индикатор присутствия объекта в конкретном рюкзаке)

при условии:

- $\sum_{i=1}^{N_M} w_{in} * X_{in} \le C_n, \forall n \in [1; N_K]$
  (ёмкость каждого рюкзака не должна быть превышена);
- $\sum_{n=1}^{N_K} X_{in} = 1, \forall i \in [1; N_M]$
  (каждый объект должен быть помещен ровно в 1 рюкзак).

Результат представим в виде матрицы X, размерностью $N_M \times N_K$, где X(i, n) = X_{in}.

2. Задача распределения вычислительной нагрузки в модульных вычислительных системах и ее представление в форме обобщенной задачи о мультипликативном рюкзаке со связями

Как было сказано раньше, задача OMP со связями возникает при распределении вычислительной нагрузки в модульных вычислительных системах. Опишем подробнее взаимосвязь этих задач.

Модульная вычислительная система представляет из себя набор модулей, соединенных коммуникационной средой. Каждый модуль может содержать одно или более процессорных ядер. Вычислительная нагрузка для подобных систем состоит из набора разделов. Каждый раздел содержит задачи, относящиеся к одной подсистеме. Все задачи одного раздела должны выполняться на одном ядре, с заданными периодами. Длительность выполнения задач зависит от типа процессора, на ядре которого она выполняется. Также задачи могут производить обмен сообщениями между собой (объем передаваемых сообщений известен). Если происходит обмен информацией между задачами, выполняемыми на разных модулях, то он вносит вклад в загрузку межмодульной сети (коммуникационной среды).

Планирование вычислений в модульных ВС происходит в несколько этапов [7]. Вначале разделы распределяются по модулям и процессорным ядрам. Затем для каждого ядра строится расписание окон — интервалов времени, в рамках которых допустимо выполнение задач соответствующих разделов. В ходе функционирования ВС, в каждом окне порядок выполнения задач осуществляется динамическим планировщиком. В настоящей работе рассматривается первый из перечисленных этапов.

Распределение разделов по модулям и процессорным ядрам выбирается исходя из следующих соображений:

- соблюдение заданных верхних ограничений на загрузку процессорных ядер;

- минимизация нагрузки на межмодульную сеть с целью сокращения задержек на передачу сообщений через сеть.

Задача распределения разделов по процессорным ядрам (и, как следствие, модулям) сводится к задаче OMP со связями между объектами следующим образом. Ядрам соответствуют рюкзаки. Разделам — заданные для упаковки объекты. В качестве ёмкостей рюкзаков выступают ограничения на загрузку соответствующих процессорных ядер. Вклад объекта в загрузку каждого из рюкзаков приравнивается к сумме длительностей задач соответствующего раздела, умноженных на их частоту. Числовая характеристика связей между объектами определяется объемом сообщений, передающихся между соответствующими разделами.

Решив полученную задачу OMP со связями, получим распределение вычислительной нагрузки в заданной системе с минимизацией трафика в сети.

3. Нахождение оценок для метода ветвей и границ

Одним из алгоритмов решения обобщенной задачи о мультипликативном рюкзаке является алгоритм, основанный на методе ветвей и границ (МВГ) и описанный в работе [2]. Решение модифицированной задачи также может быть найдено методом ветвей и границ. Для построения алгоритма необходимо определить функции вычисления верхних и нижних границ целевой функции на подобластях множества решений задачи.

Подсчет нижней границы целевой функции представляет из себя алгоритм построения допустимого решения, основанного на жадном критерии. На каждой итерации алгоритма среди всех нераспределенных объектов ищется объект с максимальной разностью между двумя его наибольшими степенями сродства и помещается в рюкзак, соответствующий максимальной степени сродства. Под степенью сродства объекта с заданным рюкзаком (f_{jn}) понимается либо отношение суммарной связи данного объекта со всеми объектами, уже распределенными в этот рюкзак, к вкладу данного объекта в заполнение заданного рюкзака, либо, если в данном рюкзаке еще нет объектов, отношение емкости заданного рюкзака к вкладу данного объекта в его заполнение.

$$f_{jn} = \begin{cases} \dfrac{C_n}{w_{jn}}, & \text{если в рюкзаке } \mathrm{K_N} \in F_j \\ & \text{нет ни одного объекта;} \\ \dfrac{\sum_{M_i \in K_n} l_{ij}}{w_{jn}}, & \text{иначе} \end{cases}$$

(здесь *j* — номер объекта, *n* — номер рюкзака)

Если на некоторой итерации используемый жадный алгоритм не может построить допустимое решение, то нижняя граница в рассматриваемом узле дерева приравнивается нулю.

Подсчет верхней границы представляет из себя поиск приближенного значения супремума целевой функции. Схема подсчета верхней границы следующая. Рассматриваем каждый рюкзак (K_n) отдельно и наполняем его любыми из нераспределенных объектов так, чтобы суммарная связь (S_n) между объектами внутри него была максимально возможной. Получим грубую верхнюю оценку:

$$\overline{U_0} = \sum_{n=1}^{N_K} S_n$$

После упаковки рюкзаков по отдельности можно выделить два подмножества исходного набора нераспределенных объектов: M^0 — множество объектов, не попавших ни в один рюкзак; M^{2+} — множество объектов, попавших в более, чем один, рюкзак. Для приближения полученной грубой оценки к супремуму целевой функции найдем объект (M_i) из множеств M^0 или M^{2+}, добавление в один из рюкзаков (или изъятие из одного рюкзака соответственно) которого приведет к максимальному изменению (D_i) суммы связей между объектами внутри рюкзаков. Тогда итоговая верхняя граница будет следующей:

$$U_2 = \overline{U_0} - D_i$$

4. Обоснование корректности построенных оценок

Рассчитываемая в рамках метода ветвей и границ нижняя граница для целевой функции является корректной в случае, если она достигается на одном из допустимых решений. Расчет этой границы в предложенном алгоритме обеспечивает ее корректность за счет того, что строит допустимое решение, на котором эта граница достигается. Если решения построено не было, нижняя граница приравнивается к 0 — минимальному значению целевой функции.

Верхняя граница для целевой функции на подобласти множества решений задачи является корректной, если значение целевой функции на любом допустимом решении из этой подобласти не превышает этой границы.

Чтобы обосновать корректность предложенной схемы расчета верхней границы целевой функции, достаточно доказать, что $U_2 - U_{opt} \geq 0$, где U_{opt} — оптимальное решение задачи.

Основная идея выполненного авторами данной работы доказательства корректности получаемой верхней границы заключена в разбиении элементов, входящих в составляющие рассматриваемой разности, на составные части и их перегруппировке в сумму неотрицательных выражений. Приведем основные выкладки данного доказательства:

1. Если $M_i \in M^{2+}$, то:

$$D_i = \sum_{K_n \in K^{2+}(i)} (S_n - \min(S_n, u_{i,n}^0)) - \max_{K_n \in K^{2+}(i)} \{S_n - \min(S_n, u_{i,n}^0)\}$$

Введем новые обозначения:

- $i_{\max}$ — номер рюкзака, на котором достигается $\max_{K_n \in K^{2+}(i)} \{S_n - \min(S_n, u_{i,n}^0)\}$ для объекта M_i;
- i_{true} — номер рюкзака, в котором находится объект M_i в оптимальном решении;
- S_n — общая стоимость объектов в рюкзаке K_n на первом шаге алгоритма;
- S_n^{opt} — общая стоимость объектов в рюкзаке K_n в оптимальном решении

Распишем U_2:

$$U_2 = \overline{U_0} - D_i = \sum_{K_n \in K^{2+}(i) \setminus \{K_{i_{\max}}\}} u_{i,n}^0 + \sum_{K_n \notin K^{2+}(i)} S_n + S_{i_{\max}};$$

1.1. Рассмотрим разность ($U_2 - U_{opt}$):

1.1.1. Если $K_{i_{\max}} = K_{i_{true}}$, то:

$$U_2 - U_{opt} = (S_{i_{\max}} - S^{opt}_{i_{\max}})$$

$$+ \Big(\sum_{K_n \in K^{2+}(i) \setminus \{K_{i_{\max}}\}} u^0_{i,n} - \sum_{K_n \in K^{2+}(i) \setminus \{K_{i_{\max}}\}} S^{opt}_n \Big)$$

$$+ \Big(\sum_{K_n \notin K^{2+}(i)} S_n - \sum_{K_n \notin K^{2+}(i)} S^{opt}_n \Big)$$

Отметим, что каждое слагаемое, взятое в скобки, не меньше 0. И значит, что разность не меньше 0: $U_2 - U_{opt} \geq 0$;

1.1.2. Если $K_{i_{\max}} \neq K_{i_{true}}$, то:

1.1.2.1. Если $K_{i_{true}} \notin K^{2+}(i)$:

$$U_2 - U_{opt} = \Big(\sum_{K_n \in K^{2+}(i) \setminus \{K_{i_{\max}}\}} u^0_{i,n} - \sum_{K_n \in K^{2+}(i) \setminus \{K_{i_{\max}}\}} S^{opt}_n \Big)$$

$$+ \Big(\sum_{K_n \notin K^{2+}(i) \setminus \{K_{i_{true}}\}} S_n - \sum_{K_n \notin K^{2+}(i) \setminus \{K_{i_{true}}\}} S^{opt}_n \Big)$$

$$+ (S_{i_{\max}} - S^{opt}_{i_{\max}}) + (S_{i_{true}} - S^{opt}_{i_{true}})$$

Отметим, что каждое слагаемое, взятое в скобки, не меньше 0 (аналогично предыдущему пункту).

И значит, что разность не меньше 0: $U_2 - U_{opt} \geq 0$;

1.1.2.2. Если $K_{i_{true}} \in K^{2+}(i)$:

$$U_2 - U_{opt} = \Big(\sum_{K_n \in K^{2+}(i) \setminus \{K_{i_{\max}} \bigcup K_{i_{true}}\}} u^0_{i,n} - \sum_{K_n \in K^{2+}(i) \setminus \{K_{i_{\max}} \bigcup K_{i_{true}}\}} S^{opt}_n \Big) + \Big(\sum_{K_n \notin K^{2+}(i)} S_n - \sum_{K_n \notin K^{2+}(i)} S^{opt}_n \Big) + (u^0_{i,i_{true}} - S^{opt}_{i_{true}} + S_{i_{\max}} - u^0_{i,i_{\max}}) + (u^0_{i,i_{\max}} - S^{opt}_{i_{\max}})$$

Дадим следующие оценки:

$$\sum_{K_n \in K^{2+}(i) \setminus \{K_{i_{\max}} \bigcup K_{i_{true}}\}} u^0_{i,n} - \sum_{K_n \in K^{2+}(i) \setminus \{K_{i_{\max}} \bigcup K_{i_{true}}\}} S^{opt}_n \geq 0;$$

$$\sum_{K_n \notin K^{2+}(i)} S_n - \sum_{K_n \notin K^{2+}(i)} S^{opt}_n \geq 0;$$

$$u^0_{i,i_{\max}} - S^{opt}_{i_{\max}} \geq 0;$$

заметим, что:

$$\begin{cases} u^0_{i,i_{true}} - S^{opt}_{i_{true}} \geq u^0_{i,i_{true}} - S_{i_{true}} \\ | u^0_{i,i_{true}} - S_{i_{true}} | \leq | S_{i_{\max}} - u^0_{i,i_{\max}} | \\ S_{i_{\max}} - u^0_{i,i_{\max}} \geq 0 \end{cases}$$

Из двух последних строк следует, что:

$$u^0_{i,i_{true}} - S_{i_{true}} + S_{i_{\max}} - u^0_{i,i_{\max}} \geq 0.$$

В силу первой строки:

$$u^0_{i,i_{true}} - S^{opt}_{i_{true}} + S_{i_{\max}} - u^0_{i,i_{\max}} \geq 0$$

С учетом указанных оценок:

$U_2 - U_{opt} \geq 0$;

2. Если $M_i \in M^0$, то:

$$D_i = \min_{K_n \in K} \{S_n - \min(S_n, u^1_{i,n})\}$$

Введем новые обозначения:

- $i_{\min}$ — номер рюкзака, на котором достигается $\min_{K_n \in K} \{S_n - \min(S_n, u^1_{i,n})\}$ для объекта M_i;
- i_{true} — номер рюкзака, в котором находится объект M_i в оптимальном решении;
- S_n — общая стоимость объектов в рюкзаке K_n на первом шаге алгоритма;
- S_n^{opt} — общая стоимость объектов в рюкзаке K_n в оптимальном решении;

2.1. Рассмотрим разность ($U_2 - U_{opt}$)

2.1.1. Если $K_{i_{\min}} = K_{i_{true}}$, то $u^1_{i,i_{\min}} = S^{opt}_{i_{true}}$ и:

$$U_2 - U_{opt} = \left(\sum_{K_n \in K \setminus \{K_{i_{\min}}\}} S_n - \sum_{K_n \in K \setminus \{K_{i_{\min}}\}} S_n^{opt}\right) + (u^1_{i,i_{\min}} - S^{opt}_{i_{true}}) \geq 0;$$

2.1.2. Если $K_{i_{\min}} \neq K_{i_{true}}$, то $u^1_{i,i_{true}} = S^{opt}_{i_{true}}$ и:

$$U_2 - U_{opt} = \left(\sum_{K_n \in K \setminus \{K_{i_{\min}} \bigcup K_{i_{true}}\}} S_n - \sum_{K_n \in K \setminus \{K_{i_{\min}} \bigcup K_{i_{true}}\}} S_n^{opt}\right) + (S_{i_{true}} - S^{opt}_{i_{true}} - \min_{K_n \in K}\{S_n - u^1_{i,n}\}) + (S_{i_{\min}} - S^{opt}_{i_{\min}});$$

Заметим, что справедливы следующие оценки:

$$S_{i_{true}} - S^{opt}_{i_{true}} \geq \min_{K_n \in K} \{S_n - u^1_{i,n}\}$$

$$\Rightarrow S_{i_{true}} - S^{opt}_{i_{true}} - \min_{K_n \in K} \{S_n - u^1_{i,n}\} \geq 0 \quad ;$$

$$S_{i_{\min}} - S^{opt}_{i_{\min}} \geq 0 ;$$

$$\sum_{K_n \in K \setminus \{K_{i_{\min}} \bigcup K_{i_{true}}\}} S_n - \sum_{K_n \in K \setminus \{K_{i_{\min}} \bigcup K_{i_{true}}\}} S^{opt}_n \geq 0 ;$$

С учетом указанных оценок:

$$U_2 - U_{opt} \geq 0 ;$$

Во всех вышерассмотренных случаях:

$U_2 - U_{opt} \geq 0$;

И значит, $U_2 = \overline{U_0} - D_i$ действительно является верхней границей.

5. Сложность предложенного алгоритма

Верхняя оценка вычислительной сложности представленного алгоритма имеет порядок $O({N_K}^{N_M} * (N_K * N_M * \log(N_M)))$, где $O({N_K}^{N_M})$ — верхняя оценка числа вершин дерева поиска, $O(N_K * N_M * \log(N_M))$ — верхняя оценка сложности вычисления границ в каждой вершине.

6. Экспериментальное исследование предложенного алгоритма

Экспериментальное исследование было проведено на двух группах наборов исходных данных. В первом случае каждый объект разрешается класть в любой рюкзак; во втором случае вводится запрет на размещение объекта в некоторые рюкзаки. Назовём это ограничение ограничением на привязку объектов к рюкзакам.

На рисунке 1 представлена зависимость среднего времени построения решения задачи от числа рюкзаков и распределяемых объектов.

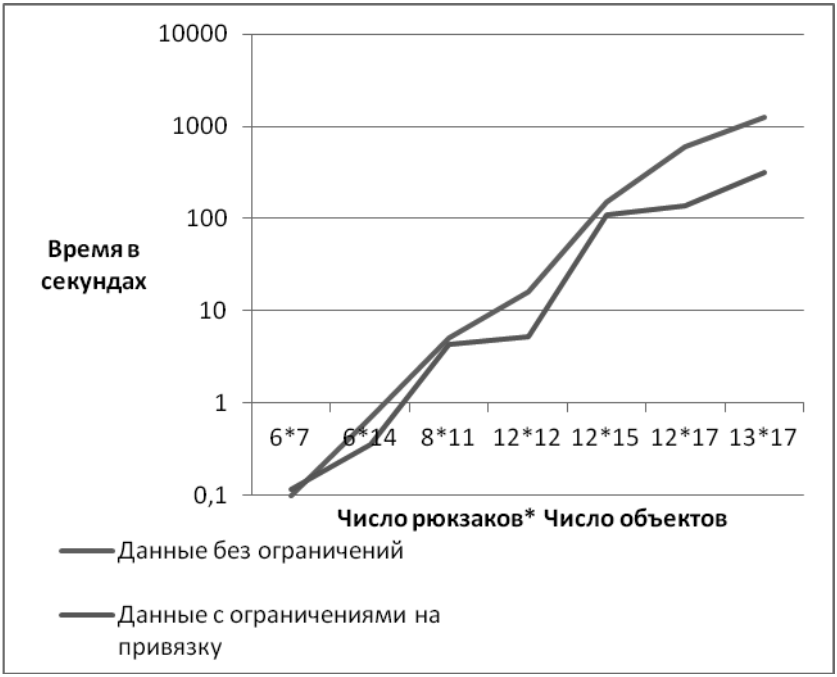

Рисунок 1. *Зависимость времени работы алгоритма от размерности задачи*

Видно, что скорость построения оптимальной упаковки объектов по рюкзакам алгоритмом, основанным на методе ветвей и границ с оценками, описанными в данной статье, на входных данных с ограничениями на привязку объектов выше, чем скорость на данных без ограничений. Это связано с тем, что ограничивая привязку объектов, мы уменьшаем число возможных решений, т.е. ведется просмотр меньшего числа узлов дерева поиска.

Отметим, что данная производительность достаточно высока и алгоритм можно применять для решения уже реальных прикладных задач. Размерность входных данных в 12 рюкзаков и 17 объектов соответствует реалистичной размерности входных данных для задачи распределения вычислительной нагрузки в модульных вычислительных системах.

7. Заключение

В данной статье представлен новый тип задачи о рюкзаке — обобщенная задача о мультипликативном рюкзаке со связями между объектами. Приведена ее формальная постановка и ее соответствие

задаче о распределении вычислительной нагрузки в вычислительных модульных системах. Описаны построение оценок для алгоритма, основанного не методе ветвей и границ, решающего данную задачу, а также основные идеи доказательства их корректности. Проведено экспериментальное исследование алгоритма, использующего описанные оценки, на двух типах входных данных. В дальнейшем планируется улучшить схему оценки верхней границы для алгоритма, основанного на методе ветвей и границ, что позволит более эффективно выполнять отсечение бесперспективных подобластей множества решений и ускорить работу алгоритма.

Литература

1. Balashov V.V., Balakhanov V.A., Kostenko V.A. Scheduling of computational tasks in switched network-based IMA systems // Proc. International Conference on Engineering and Applied Sciences Optimization. – National Technical University of Athens (NTUA) Athens, Greece, 2014. – P. 1001–1014.
2. Silvano Martello, Paolo Toth. Knapsack problems: algorithms and computer implementations. John Wiley & Sons, 1990.
3. Michel Gendreau, Jean-Yves Potvin. Handbook of metaheuristics (second edition). Springer, 2010.
4. Смирнов А. М. О задаче упаковки в контейнеры // Успехи математических наук, 1991, т. 46, вып. 4 (280), 173–174.
5. M. Dawande, J. Kalagnanam. Approximation algorithms for the multiple knapsack problem with assignment restrictions // J. Combinatorial Optimization, 2000, Vol. 4 (2), P. 171–186.
6. Шукалов А.В., Парамонов П.П., Книга Е.В., Жаринов И.О. Принципы построения вычислительных компонентов систем интегрированной модульной авионики // Информационно-управляющие системы, 2014, №5, с.64-71.
7. Balashov V. V., Balakhanov V. A., Kostenko V. A. Scheduling of computational tasks in switched network-based ima systems // Proc. International Conference on Engineering and Applied Sciences Optimization. — National Technical University of Athens (NTUA) Athens, Greece, 2014. — P. 1001–1014

Малышев Н.С., Волканов Д.Ю.

Применение поиска клонов исходного кода в задаче обнаружения ошибок в программе

Введение

При разработке программ в исходном коде регулярно появляются фрагменты кода, содержащие ошибки или уязвимости. При их эксплуатации может наблюдаться непредсказуемое поведение программы или получение несанкционированного доступа к ресурсам и функциям системы. Для обнаружения подобных фрагментов используется статический анализ, позволяющий локализовать в тексте программы проблемы и выяснить их тип.

Кроме того, в исходном коде программы могут присутствовать так называемые «клоны» или «дубликаты» кода, то есть такие фрагменты исходного кода, которые незначительно отличаются друг от друга и обладают схожим функционалом. Наличие клонов в исходном коде приводит к усложнению модификации, разработки, поддержки программы, реорганизации исходного кода и увеличению количества ошибок и уязвимостей в программе, поскольку если какая-то проблема присутствует в некотором фрагменте, то может присутствовать и в его клоне [1, 2].

Также, в силу активного использования систем управления версиями, наличие фрагментов исходного кода, содержащих ошибки или уязвимости, в одной из версий программы в репозитории означает, что во всех непосредственных и прямых потомках этой версии могут присутствовать указанные фрагменты и их клоны.

В случае обнаружения ошибки в старой версии программы необходимо выяснить в каких более новых версиях она присутствует. Для решения данной задачи можно прибегнуть к применению статического анализа ко всем версиям, в которых производится поиск ошибок. В данной работе был предложен другой подход, подразумевающий обнаружение ошибок и уязвимостей в нескольких версиях посредством статического анализа и последующий поиск клонов кода обнаруженных фрагментов в оставшейся части проверяемых версий. Было высказано предположение, что подобный метод позволит не только обнаруживать множество ошибок, аналогичное поверсионному статическому анализу, но и производить анализ быстрее, начиная с некоторого объёма исходного кода.

1. Постановка задачи анализа версий программы на предмет клонирования фрагментов исходного кода, содержащих ошибки и уязвимости

Вначале определим понятие клона исходного кода. Разумно рассматривать клоны с точки зрения возможности их появления в процессе написания программы. Процесс внесения клонов по своей сути обратен процессу удаления клонов из текста программы при реорганизации (или рефакторинге - изменении внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы) кода. Для того, чтобы избавить исходный код от клонов, применяется две операции [3, 4]:

- выделение общего кода в новую функцию или метод класса;
- перемещение общего метода классов в их базовый класс.

Минимальным фрагментом кода, который может быть выделен в функцию или метод, является последовательность операторов. Однако из-за существования составных и вложенных операторов можно неоднозначно истолковать понятие последовательности операторов, поэтому рассматриваться будут только фрагменты кода, представляющие собой непрерывные последовательности операторов.

Выделение общего кода в новую функцию допускает наиболее широкие отличия между фрагментами кода, а именно – замены одних синтаксических единиц на другие. Синтаксической единицей называется такой фрагмент исходного кода, к которому могут быть применены синтаксические правила языка программирования. Например, константы и подвыражения являются частными случаями синтаксических единиц, “x” или “x / 3” являются синтаксическими единицами, а “x /” таковой не является. Соответственно, можно ввести следующее определение клонов кода.

Определение 1. Два фрагмента исходного кода *F1* и *F2* программы являются клонами кода удалённости *d*, если они представляют собой непрерывные последовательности операторов равного размера (то есть из одинакового числа операторов) и один может быть получен из другого путём замены одних синтаксических единиц на другие и справедливо равенство $\rho(F1, F2) = d$, где $\rho(F1, F2)$ – некоторая функция расстояния между фрагментами кода.

Данное определение позволяет точно понять, какие фрагменты исходного кода можно сравнивать, поскольку определить, является ли фрагмент синтаксической единицей можно при помощи грамматики языка программирования, а также позволяет ограничивать сверху множество тех фрагментов исходного кода, которые считаются клонами, благодаря введённой функции расстояния.

Репозиторий проекта в системе управления версиями можно описать, как множество версий, содержащих в свою очередь множество файлов с исходным кодом и других файлов, сопутствующих разработке программ, а также связей между версиями, благодаря которым для каждой версии можно выяснить множество версий, являющихся её прямыми потомками. Даже если рассматривается не проект в некоторой системе управления версиями, а просто набор версий программы, вышеописанная структура множества версий обязательна для корректного анализа на наличие ошибок.

Рассматриваться может как множество всех версий программы, так и его подмножество, необходимым является лишь наличие в анализируемом наборе версий тех версий программы, в которых были обнаружены ошибки и уязвимости. Обозначим выбранное для анализа множество версий, как R.

Подмножество P множества R указывается в явном виде и используется для инициализации процесса поиска ошибок – именно в версиях программы из этого множества будет произведён поиск фрагментов исходного кода, содержащих ошибки, клоны кода которых будут обнаружены в других версиях. Версии программы из множества P должны содержать ошибки, поскольку в случае отсутствия ошибок в P поиск клонов является бессмысленным.

Результатом статического анализа исходного кода программы является набор номеров строк в файлах с текстом программы, содержащих ошибки с точки зрения анализатора. В связи с этим, вместе со строкой, на которую указал анализатор необходимо рассматривать и её контекст, иначе клонами можно было бы считать слишком большое количество фрагментов кода.

Таким образом для поиска ошибок в исходном коде программы с использованием статического анализа и методик поиска клонов кода необходимо разработать метод анализа множества версий R программы на предмет клонирования фрагментов исходного кода, содержащих ошибки и уязвимости, обнаруживающий в множестве версий R \ P все фрагменты исходного кода не больше заданного размера, являющиеся клонами найденных при статическом анализе версий из множества P фрагментов с точки зрения выбранного определения клонов кода.

2. Способы представления исходного кода

Введения определения 1 недостаточно для поиска клонов кода, поскольку необходимо выбрать то представление исходного кода, в котором будут сравниваться фрагменты – то, с которым будет работать алгоритм поиска.

Во многих работах способы представления исходного кода делятся на следующие категории [1, 3, 5]:

- последовательность символов, то есть текст [6, 15];
- последовательность лексем [2, 6, 7];
- абстрактное синтаксическое дерево [8, 9];
- числовые характеристики кода [10, 11];
- граф зависимостей программы по данным и управлению [12, 13].

Способ представления выбирался согласно следующим критериям:

1. способ представления исходного кода позволяет обнаруживать клоны кода, удовлетворяющие определению 1;
2. для указанного способа представления кода существуют доступные средства, способные сформировать это представление для произвольного фрагмента кода.

Очевидно, что представление исходного кода в виде текста не позволит обнаруживать клоны, соответствующие определению 1, поскольку не подразумевает отличий между клонами, либо же отличия могут нарушать синтаксис языка программирования.

Представление в виде последовательности лексем позволяет обнаруживать клоны, отличающиеся в идентификаторах, константах и в произвольных лексемах, но не позволяет обнаруживать клоны, удовлетворяющие определению 1, так как на уровне последовательности лексем невозможно выделить синтаксические единицы исходного кода.

Абстрактное синтаксическое дерево (АСД) – это конечное, размеченное, ориентированное дерево, где внутренним вершинам соответствуют операторы языка программирования, а листьям – их операнды. Любая синтаксическая единица может быть представлена в виде синтаксического дерева и любое синтаксическое дерево однозначно соответствует некоторой синтаксической единице. Однако не для всех языков программирования можно получить подобное представление для произвольного фрагмента кода.

Подход, использующий числовые характеристики, предполагает вычисление для анализируемых фрагментов (функций, классов или целых файлов) числовых характеристик, таких как количество символов, строк, лексем, обращений к файлам, операций ввода/вывода, вызовов функций. Использование только числовых характеристик не подходит для поиска клонов, отвечающих определению 1.

Граф зависимостей программы (ГЗП) – это абстрактное представление программы, хранящее информацию о зависимостях по данным и управлению. Клонами с точки зрения ГЗП являются семантически схожие фрагменты кода. Использование ГЗП позволяет находить клоны, отличающиеся пробелами, комментариями, идентификаторами, константами, значениями переменных и вставкой

или удалением некоторого числа строк, но не предполагает отличий в подвыражениях.

Итоги обзора способов представления исходного кода программы представлены в таблице 1, “±” означает, что не для всех языков программирования критерий выполняется.

	Критерий 1	Критерий 2
Текст	-	+
Лексемы	-	+
АСД	+	±
Характеристики	-	+
ГЗП	-	±

Таблица 1. *Обзор способов представления исходного кода программы*

В результате проведённого обзора абстрактные синтаксические деревья были выбраны в качестве представления исходного кода программы.

3. Расстояние по антиунификации

В качестве расстояния предлагается использовать так называемое расстояние по антиунификации [1, 13], инвариантное относительно способа представления фрагмента исходного кода в виде абстрактного синтаксического дерева, отражающее структуру фрагментов и чувствительное к замене больших поддеревьев или изменению порядка поддеревьев в деревьях, соответствующих фрагментам. Это расстояние вычисляется благодаря построению наиболее специализированного общего шаблона.

Общим шаблоном двух деревьев T_1 и T_2 называется такое дерево, в некоторых листьях которого расположены специальные метки-заполнители, путём подстановки вместо которых некоторых поддеревьев можно получить и дерево T_1, и дерево T_2.

Наиболее специализированным общим шаблоном (НСОШ) *MSCP* двух деревьев T_1 и T_2 называется такой общий шаблон этих деревьев, что любой другой общий шаблон T_1 и T_2 будет являться также и шаблоном *MSCP*.

В качестве размера дерева предлагается использовать количество листьев с полноценными операндами. Такое расстояние не зависит от способа представления дерева в виде АСД.

Тогда расстоянием между фрагментами F_1 и F_2 назовём число, равное суммарному размеру подставляемых в их НСОШ поддеревьев для получения T_1 и T_2, из которого вычтено число меток-заполнителей в НСОШ.

4. Алгоритм поиска клонов кода фрагментов, содержащих ошибки и уязвимости

Опишем предлагаемый алгоритм поиска клонов кода фрагментов, содержащих ошибки и уязвимости, по множеству версий проекта. При начале работы алгоритма известны только анализируемое множество версий *R* и его подмножество *P*, при помощи которого производится инициализация.

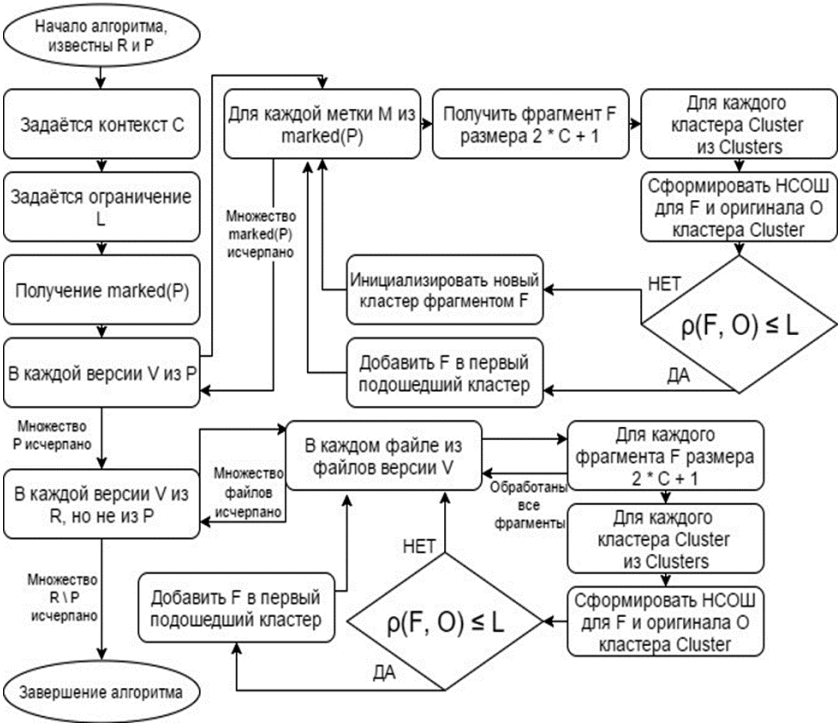

Рисунок 1. *Схема работы алгоритма*

На рисунке 1 приведена схема работы вышеописанного алгоритма.

На вход алгоритм принимает множество версий *R* и его подмножество *P*, а также размер обрабатываемых алгоритмом фрагментов исходного кода *C* (контекст) и ограничение на расстояние между фрагментами *L*. После этого производится статический анализ версий из *P*, результатом которого является множество меток *marked(P)*, в котором каждой обнаруженной ошибке соответствует информация о её положении в исходном коде в виде метки *M*. Затем, для каждой

обнаруженной таким образом ошибки производится проверка принадлежности содержащего её фрагмента кода к непересекающимся множествам клонов кода (кластерам) при помощи условия $\rho(F, O) \leq L$. Здесь вычисляется расстояние между обрабатываемым фрагментом F и первым элементом кластера (оригиналом кластера) O. В том случае, если фрагмент не принадлежит ни одному из кластеров, он инициализирует новый кластер. После завершения инициализации в оставшихся версиях (то есть версиях из $R \backslash P$) производится проверка принадлежности фрагментов F размера $2*C+1$ в файлах исходного кода этих версий уже существующим кластерам, согласно условию $\rho(F, O) \leq L$. Однако, в случае, когда фрагмент не подходит ни к одному кластеру, он не инициализирует новый кластер, а пропускается. Алгоритм завершает свою работу после обработки всех фрагментов исходного кода размера $2*C+1$ в версиях из $R \backslash P$. Результатом работы алгоритма является набор непересекающихся множеств клонов исходного кода, в каждом из которых элементы являются клонами оригинала этого множества, обнаруженного статическим анализатором на этапе инициализации.

5. Реализация алгоритма

Алгоритм был реализован на C++ и Python в виде программного средства для анализа репозиториев проектов на C99 в системе управления версиями Git. Схема работы реализации представлена на рисунке 2.

В качестве средства статического анализа после аналитического обзора 20 средств для этапа инициализации была выбрана утилита CppCheck, как единственная одновременно удовлетворяющая следующим критериям:

- работа только с файлами исходного кода (не требует внесения аннотаций в исходный код перед применением);
- доступность для использования в рамках исследования;
- вывод результата в виде текста в файл;
- управление из командной строки;
- поиск наиболее частых ошибок, встречающихся в исходном коде, таких как деление на ноль, выход за границу массива и переполнение при арифметических операциях [21].

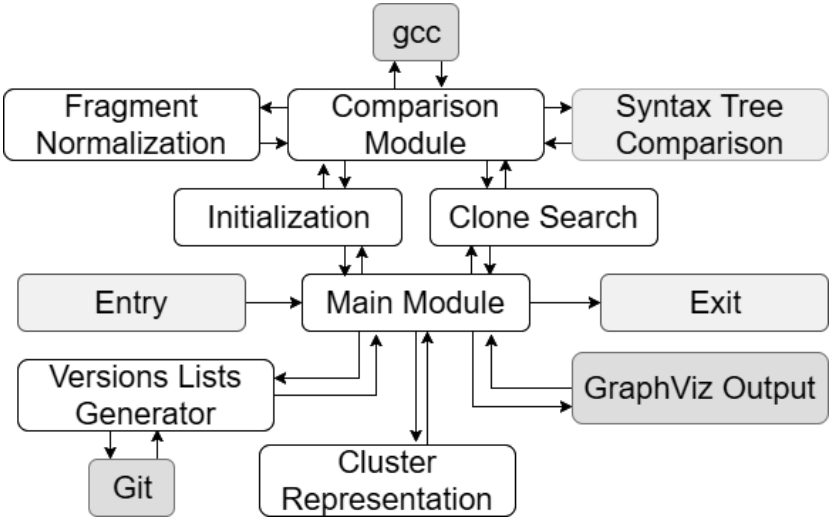

Рисунок 2. *Схема взаимодействия модулей программы*

Разработанное средство состоит из следующих модулей:

- *Main Module* запускает процесс формирования списков версий P и R\P, инициализации, поиска клонов кода и вывода результата.
- *Versions Lists Generator* подготавливает внутрипрограммное представление множеств версий P и R \ P при помощи функций системы управления версиями *Git*.
- *Initialization* производит формирования множеств клонов кода по версиям из P в соответствии с работой алгоритма.
- *Clone Search* производит поиск клонов кода в версиях из R \ P в соответствии с работой алгоритма.
- *Comparison Module* готовит сравниваемые фрагменты к сравнению модулем *Syntax Tree Comparison*, обращаясь к модулю *Fragment Normalization* и производя предкомпиляцию результата при помощи *gcc*.
- *Syntax Tree Comparison* формирует синтаксические деревья сравниваемых фрагментов, их наиболее специализированный общий шаблон, вычисляет между фрагментами расстояние по антиунификации и возвращает результат сравнения модулю *Comparison Module.*
- *Cluster Representation* формирует представление множеств в формате, предназначенном для использования *GraphViz*, в модуле *GraphViz Output*, для визуализации результата анализа.

6. Экспериментальное исследование реализации алгоритма

Целью исследования является выяснение способности предложенного алгоритма обнаруживать ошибки в исходном коде, для чего было проведено сравнения количества и качества результата работы с поверсионным применением утилиты CppCheck, а также оценка зависимости времени выполнения анализа от объёма входных данных.

Сравнение времён выполнения анализа напрямую нецелесообразно, поскольку модуль сравнения фрагментов как синтаксических деревьев был написан на Python в силу удобства этого языка для реализации нужного функционала, и известно, что реализация алгоритмов на Python значительно уступает в быстродействии аналогам на C++ [14].

Эксперименты были проведены со специально подготовленным репозиторием, содержащим ошибки, репозиторием с исходным кодом самой исследуемой реализации предложенного алгоритма и репозиториями открытых проектов Cranium [16], CCWT [17], TinyJPEG [18], LibGOST15 [19] и CINIParser [20], загруженных с ресурсва GitHub.

Было выяснено, что время анализа версий для каждого отдельного проекта реализацией алгоритма линейно зависит от объёма исходного кода, измеряемого в количестве строк, в то время, как зависимость времени поверсионного применения CppCheck зависит от объёма исходного кода экспоненциально. Из этого следует, что для каждого проекта существует значение объёма исходного кода, начиная с которого CppCheck будет производить поиск ошибок медленнее.

На рисунке 3 изображены графики зависимостей времени выполнения анализа от объёма исходного кода для одного из проекта, участвовавших в эксперименте. Красный график – реализация алгоритма, синий – CppCheck.

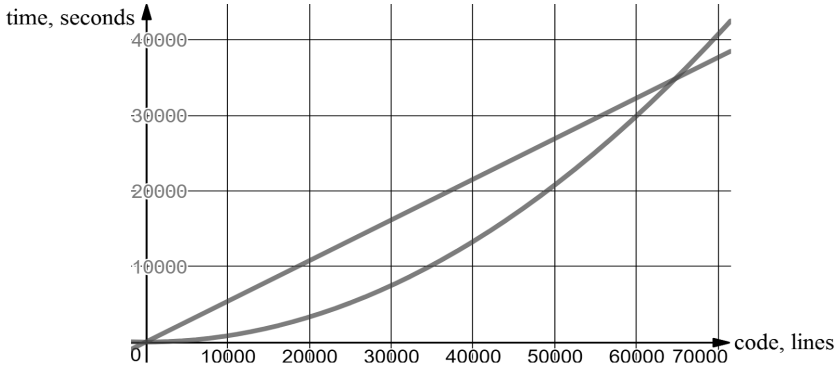

Рисунок 3. *Графики зависимостей времени анализа от объёма исходного кода*

Репозиторий	Число версий	CppCheck	Алгоритм
Подготовленный репозиторий	2	13	13 + 2д.
Подготовленный репозиторий	5	31	31 + 2д.
Подготовленный репозиторий	10	41	41 + 2д.
Реализация ВКР	2	4	4
Реализация ВКР	7	14	14
Cranium	2	28	28
Cranium	5	69	69
CCWT	2	2	2
CCWT	5	5	5
CCWT	10	10	10
Tiny JPEG	7	1	1
Tiny JPEG	10	4	4
LibGOST 15	2	2	2 + 6 л-п.

LibGOST 15	5	5	5 + 10 л-п.
LibGOST 15	7	7	7 + 22 л-п.
C-INI-Parser	2	4	4
C-INI-Parser	6	12	12

Таблица 2. *Результаты экспериментального исследования*

В таблице 2 изображены количества найденных CppCheck и реализацией алгоритма ошибок. Как видно, на подготовленном репозитории алгоритм обнаружил на две ошибки больше, а именно деление на ноль и выход за границы массива в том случае, когда вместо константы используется переменная. На LibGOST15 в результатах работы алгоритма наблюдалось большое количество ошибок первого рода (то есть фрагменты исходного кода, попавшие в множество содержащих ошибки фрагментов, в действительности ошибок не содержали). Это связано с тем, что при инициализации, в силу особенностей исходного кода LibGOST15, обнаружены были фрагменты исходного кода размера 1, содержащие оператор return, из-за чего очень большое число фрагментов могло считаться клонами. В тоже время, благодаря удачно подобранным параметрам алгоритм не пропустил ни одной ошибки, найденной CppCheck, то есть в результатах анализа отсутствовали ошибки второго рода.

Заключение

В данной статье рассмотрена проблематика определения клонов кода, способов представления исходного кода, приведено описание алгоритма поиска клонов кода фрагментов, содержащих ошибки и уязвимости, в виде блок-схемы и реализация этого алгоритма для проектов на C99.

Экспериментальное исследование данного алгоритма было проведено на подготовленном репозитории, исходном коде самой реализации и нескольких открытых проектах. Оно показало возможность превзойти скорость анализа версий проекта по сравнению с поверсионным применением статического анализатора, начиная с некоторого объёма исходного кода, сохраняя количество правильных результатов поиска.

В дальнейшем планируется исследование возможностей улучшить быстродействия алгоритма и улучшения его способности находить ошибки и уязвимости в исходном коде.

Литература

1. Bulychev P., Minea M. An evaluation of duplicate code detection using anti-unification. Proceedings of 3rd International Workshop on Software Clones, 2009, IESE-Report; 038.09/E. Pp. 22-27.
2. Ахин М. Х., Ицыксон В. М. Обнаружение клонов исходного кода: теория и практика //Системное программирование. – 2010. – Т. 5. – №. 1. – С. 145-163.
3. Roy C. K., Cordy J. R. A survey on software clone detection research //Queen's School of Computing TR. – 2007. – Т. 541. – №. 115. – Pp. 64-68.
4. Фаулер М. Рефакторинг. Улучшение существующего кода //СПб.: Символ-Плюс,-2008.-423 с. – 2009.
5. Murakami H. et al. Gapped code clone detection with lightweight source code analysis //Program Comprehension (ICPC), 2013 IEEE 21st International Conference on. – IEEE, 2013. – Pp. 93-102.
6. Baker B. S. On finding duplication and near-duplication in large software systems //Reverse Engineering, 1995., Proceedings of 2nd Working Conference on. – IEEE, 1995. – Pp. 86-95.
7. Li Z. et al. CP-Miner: Finding copy-paste and related bugs in large-scale software code //IEEE Transactions on software Engineering. – 2006. – Т. 32. – №. 3. – Pp. 176-192.
8. Зельцер Н. Г. Поиск повторяющихся фрагментов исходного кода при автоматическом рефакторинге //Труды Института системного программирования РАН. – 2013. – Т. 25.
9. Baxter I. D. et al. Clone detection using abstract syntax trees //Software Maintenance, 1998. Proceedings., International Conference on. – IEEE, 1998. – Pp. 368-377.
10. Mayrand J., Leblanc C., Merlo E. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics //icsm. – 1996. – Т. 96. – P. 244.
11. Kontogiannis K. A. et al. Pattern matching for clone and concept detection //Reverse engineering. – Springer US, 1996. – Pp. 77-108.
12. Саргсян С. и др. Масштабируемый инструмент поиска клонов кода на основе семантического анализа программ //Труды Института системного программирования РАН. – 2015. – Т. 27. – №. 1.
13. Liu C. et al. GPLAG: detection of software plagiarism by program

dependence graph analysis //Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. – ACM, 2006. – Pp. 872-881.
14. The Computer Language Benchmark Game: Python 3 vs C++ g++ - [Электронный ресурс]. – Электрон. Дан. – URL: http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=python3&lang2=gpp/ (дата обращения: 09.05.2017)
15. Ducasse S., Rieger M., Demeyer S. A language independent approach for detecting duplicated code //Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on. – IEEE, 1999. – Pp. 109-118.
16. GitHub – 100/Cranium – [Электронный ресурс]. – Электрон. дан. – URL: https://github.com/100/Cranium/ (дата обращения: 09.05.2017)
17. GitHub - Lichtso/CCWT – [Электронный ресурс]. – Электрон. дан. - URL: https://github.com/Lichtso/CCWT/tree/master/ (дата обращения: 09.05.2017)
18. GitHub - serge-rgb/TinyJPEG – [Электронный ресурс]. – Электрон. дан. - URL: https://github.com/serge-rgb/TinyJPEG/ (дата обращения: 09.05.2017)
19. GitHub - aprelev/libgost15 – [Электронный ресурс]. – Электрон. дан - URL: https://github.com/aprelev/libgost15/ (дата обращения: 09.05.2017)
20. GitHub - taka-wang/c-ini-parser – [Электронный ресурс] – Электрон. дан – URL: https://github.com/taka-wang/c-ini-parser/ (дата обращения: 09.05.2017)
21. Common C Programming Errors – [Электронный ресурс] – Электрон. дан. – URL: http://www.drpaulcarter.com/cs/common-c-errors.php#2.7/ (дата обращения: 09.05.2017)

Горохов О.Е., Петровский М.И., Машечкин И.В.

Применение сверточных нейронных сетей для задач обнаружения аномалий в работе пользователя с текстовыми данными

Введение

В современном мире возрастает интерес к решению задачи обнаружения аномалий. Суть этой проблемы заключается в выявлении данных, которые представляют собой отклонение от ожидаемого поведения [1]. Таким образом, для решения задач обнаружения аномалий мы выделяем некоторый критерий, определяющий стандартное поведение. Если данные соответствуют этому критерию, то мы будем называть их нормальными. В противном случае данные считаются аномальными.

Задача обнаружения аномалий имеет широкое применение в различных областях развития общества в настоящее время [1]. Стоит отметить, что особый интерес представляет решение задачи обнаружения аномалий при работе пользователя с текстовыми данными. Это связано с тем, что большинство информации, с которой взаимодействует пользователь, представляет собой именно текст.

Хотя анализ текстовых данных с точки зрения обнаружения аномалий в работе пользователя является чрезвычайно важной областью, на данный момент существует довольно-таки мало методов, ориентированных на данную область. Дело в том, что текстовые данные являются объектами весьма сложной структуры. Кроме того, существует проблема многозначности слов, которая придает ряд трудностей, связанных с выявлением аномалий в таких данных. Таким образом, значение каждого слова должно рассматриваться в рамках контекста, в котором это слово употреблено. Поэтому различные меры сходства документов могут терять свою надежность, вследствие чего стандартные подходы обнаружения аномалий не позволяют получить достаточно хороших результатов при анализе содержимого текстовых данных [2]. Следовательно, возникает задача обнаружения аномалий в текстовом контенте, т.е. в содержимом текстовых файлов, с которыми взаимодействует пользователь. Именно этой теме и посвящена данная работа.

Стоит отметить, что на данный момент для задачи обнаружения аномалий в работе пользователя с текстовыми данными применяются классические методы одноклассовой классификации, а также методы, основанные на тематическом моделировании [3,4]. Однако в последнее

время стало популярным применение сверточных нейронных сетей (CNN) при анализе текстов. Но в существующих работах по схожим тематикам [5,6] сверточные нейронные сети применяются лишь для решения задач многоклассовой классификации, что, как правило, обусловлено особенностями целевой функции. Поэтому в рамках данной работы предлагается провести некоторую адаптацию существующих методов классификации, построенных на основании сверточных нейронных сетей, к решению задачи обнаружения аномалий.

1. Анализ связанных работ

Исходя из анализа существующих работ по схожей тематике, можно видеть, что в общем случае схема выявления аномалий состоит из этапов, отраженных на рисунке 1.

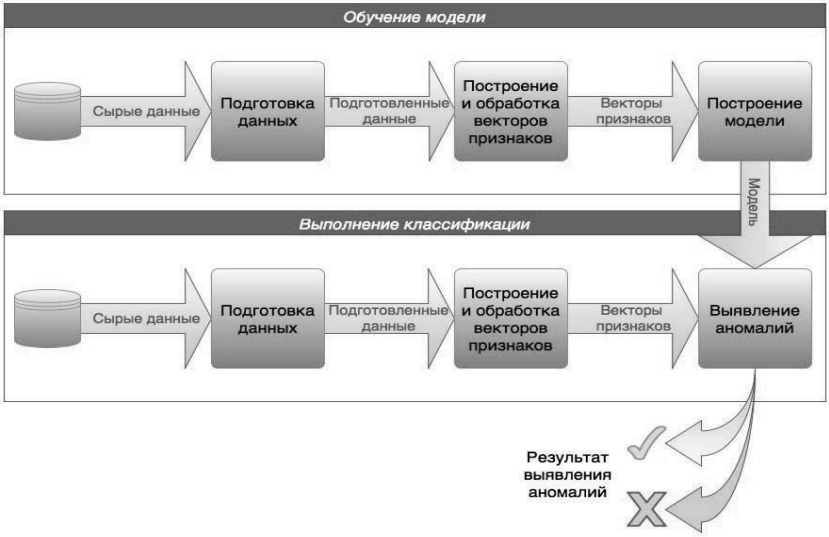

Рисунок 1. *Общая схема выявления аномальной работы пользователей*

Заметим, что анализ существующих статей ([1], [3]-[7]) показывает, что этапы подготовки данных, а также построения признаков документа в своей реализации учитывают только специфику данных, с которыми мы работаем (в нашем случае это текстовые данные). На этапе же построения и применения модели необходимо

принять во внимание непосредственные особенности поставленной задачи обнаружения аномалий.

На первом этапе, этапе обработки данных, мы будем применять метод выделения ключевых слов. Его описание можно найти в работе [3]. В качестве результата работы этого метода может стать либо последовательность целых слов из текста, либо лишь базовые словоформы.

Для вычисления векторов признаков предлагается рассмотреть два основных подхода. Классические методы на данной стадии используют частотные характеристики термов (т.н. веса термов) [3,4]. Однако данные подходы не учитывают порядок слов в документе, что является важным при работе с CNN.

В данной работе предлагается использовать альтернативный метод, основанный на представлении каждого терма в документе в виде некоторого вектора признаков. В самом простом случае терм представляется одним числом - его идентификатором во внутреннем словаре. Также стоит отметить, что данный подход используется в работах, описывающих анализ текстовых данных с использованием CNN [5,6].

Наконец, на последней стадии мы должны сформировать модель. Для этого, как правило, используется стандартная методика сведения задачи обнаружения аномалий к задаче классификации. Однако существуют и другие методы построения модели, описанные в [1].

В настоящее время для выявления аномалий в работе пользователя с текстовыми данными используются такие классические методы, как одноклассовый метод опорных векторов тематическое моделирование. Данные методы рассмотрены в работах [3] и [4].

Кроме того, стоит отметить, что в последнее время наблюдается тенденция, заключающаяся в использовании нейронных сетей в задаче обнаружения аномалий. Классическим примером в данной области является архитектура репликаторных сетей, описанная в работе [8].

Также в [5,6] рассматривается использование CNN в смежных задачах анализа текстовых данных. Однако предлагаемое в данных работах решение позволяет произвести бинарную классификацию текстовых данных. В нашем случае предлагается развитие идей применения нейронных сетей в обработке текстовых данных путем адаптации данных решений для задачи обнаружений аномалий в работе пользователя. Ниже приведено описание реализованной архитектуры нейронной сети.

2. Сверточные нейронные сети в задаче классификации текстовых данных

Сверточные нейронные сети широко применяются в компьютерном зрении. Однако было замечено, что они дают также довольно хороший результат в анализе естественного языка [5].

Стандартная сверточная сеть представляет собой многослойную нейронную сеть, где один или несколько скрытых слоев являются сверточными, т.е. позволяют выделить признаки из входных данных посредством применения свертки для некоторого фильтра.

Предложенный вариант архитектуры сверточной нейронной сети основывается на работах [5,6].

2.1 Общая архитектура

Приведем краткое описание предлагаемой архитектуры нейронной сети.

Рассмотрим некоторую обучающую выборку, состоящую из *m* документов. Как уже было сказано ранее, в качестве признаков для документа мы будем использовать последовательность векторов, соответствующих каждому терму в документе. Каждый документ в таком случае может быть представлен вектором признаков, описываемым формулой (1).

$$x_{1:n} = x_1 \oplus x_2 \oplus ... \oplus x_n, \quad (1)$$

Где $x_j \in R^k (j = 0,...n)$ – вектор, соответствующий *j*-му терму,

$k \in N$ – фиксированный размер признака,

$\oplus$ – оператор конкатенации. В общем случае, $x_{i:i+j}$ обозначает конкатенацию векторов $x_i, x_{i+1},..., x_{i+j}$.

Функционирование классической сверточной нейронной сети основывается на двух основных операциях: свертка и субдискретизация.

Операция свертки подразумевает преобразование последовательной группы признаков по определенному правилу, которое будет описано далее.

Далее считаем, что $h \in N$ – размер окна, т.е. длина подпоследовательности признаков, к которой будет применяться один фильтр. Этот размер задается на этапе формирования архитектуры сети. Теперь рассмотрим некоторый фильтр – вектор $w \in R^{h,k}$. Этот фильтр будет применен к окну, состоящему из *h* термов. В результате будет получен некий признак. При этом под применением фильтра к окну мы

будем подразумевать формулу (2), которая описывает применение фильтра w к окну $x_{i:i+h-1}$.

$$c_i = f(w * x_{i:i+h-1} + b), \quad (2)$$

где $b \in R$ – некоторое смещение,
f – нелинейная функция активации.

Таким образом, применяя данный фильтр к одному окну, мы получаем абстрактный признак c_i.

Теперь применим указанный фильтр w ко всем возможным окнам из последовательных признаков в документе**.** Таким образом, будет составлена так называемая карта признаков, описываемая формулой (3).

$$c = c_1, c_2, ...c_{n-h+1}, \quad (3)$$

где $c_i \in R$ – признак, выделенный из $x_{i:i+h-1}$ путем применения фильтра w.

После операции свертки выполняется операция субдискретизации (пулинга). Она реализует нелинейное сжатие карты признаков. Как правило, здесь используется функция максимума. В этом случае операция пулинга будет выделять наиболее значимый признак из получившейся карты. Это позволит для каждого фильтра в отдельности определить только один самый важный признак.

Таким образом, каждому отдельному фильтру будет соответствовать лишь один самый значимый элемент из карты признаков. В связи с этим, для определения нескольких признаков из документа необходимо использовать множество фильтров разной длины.

В самом общем случае в сверточной сети используется последовательное чередование пар свертка-субдискретизация. В рассматриваемой архитектуре была использована одна такая пара.

На последнем слое сети необходимо произвести некоторое отображение пространства признаков в пространство меньшей размерности для выполнения классификации.

В работах [5,6] для анализа текстовых данных предлагается архитектура, изображенная на рисунке 2 и описанная ниже.

Рассматриваемая архитектура нейронной сети состоит из слоя понижения размерности пространства входных данных, сверточного слоя, слоя субдискретизации, а также выходного слоя. Слой свертки, субдискретизации, а также выходной слой проиллюстрированы на рисунке 2.

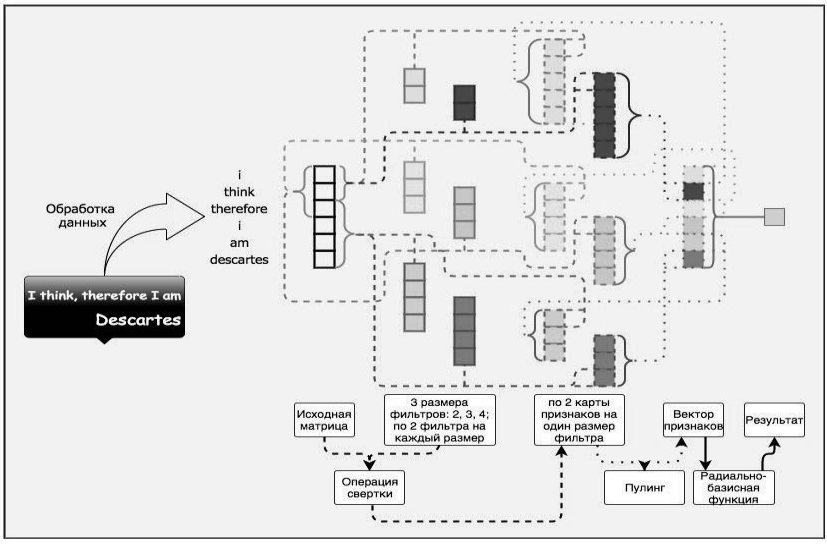

Рисунок 2. *Рассматриваемая архитектура нейронной сети*

2.2 Предлагаемая модель

В предложенном решении в роли значений, соответствующих очередному терму, выступает скалярный порядковый номер терма во внутреннем словаре (т.е. k=1). В связи с этим, в качестве первого слоя нейронной сети предлагается использовать алгоритм, который будет производить сжатие посредством объединения в одни группы термов, индексы которых имеют одинаковый остаток от деления на число – размер итогового пространства обработанных данных. Т.е. если в словаре содержится 10 термов, и необходимо «сжать» этот словарь до 5 элементов, мы объединим индексы в 5 следующих групп: [[0, 5], [1, 6], [2,7], [3, 8], [4, 9]]. Далее идет сверточный слой. Он представляет собой совокупность сверточных «подслоев», каждый из которых использует фильтр определенного размера. Для каждого такого «подслоя» использован «выпрямитель» в качестве функции активации. Все веса на этом слое инициализируются на основании усеченного нормального распределения. Смещения же инициализируются константно. Эти начальные предположения основаны на работе [5]. Вслед за каждым сверточным «подслоем» следует слой, выполняющий сжатие карты признаков путем применения операции субдискретизации. Как уже было сказано ранее, в текущей архитектуре нейронной сети используется выбор максимального значения. Сверточный слой и слой пулинга используют несколько фильтров для окон различных размеров, что позволяет получить вектор признаков. После этого, выходы всех

«подслоев», осуществляющих субдискретизацию, объединяются в один вектор. Таким образом, на этом этапе мы имеем некоторый вектор признаков, на основании которого и будет происходить выявление аномалий в работе пользователя. Далее идет финальный слой, который на основании построенных признаков позволяет произвести выявление аномальных данных.

2.3 Регуляризация и определение целевой функции

Для регуляризации на финальном слое в работах [5,6] предлагается использовать так называемый алгоритм стохастического отсечения связей (dropout). Его суть заключается в следующем. Выбирается множество синаптических связей SC, к которому будет применено отсечение. Затем каждой связи $s_i \in SC$ мы приписываем определенную вероятность p – вероятность того, что s_i будет удалена при обучении (т.е. синаптический вес соответствующей связи будет равен нулю). Этот алгоритм позволяет избежать коадаптации нейронов скрытого слоя в процессе обучения, поскольку некоторые связи будут случайно разорваны в процессе обучения. Пусть на выходе сверточной части сети был получен так называемый вектор сверточных признаков $c = (c_1,...c_m)$. Тогда на входе финального слоя вместо использования в качестве вектора признаков $c = (c_1,...c_m)$ и вектора весов $w = (w_1,...,w_m)$ мы будем использовать соответственно вектора c' и w', вычисляемые по формулам (4) и (5) соответственно.

$$c' = c \circ r, \quad (4)$$

$$w' = w \circ r, \quad (5)$$

где $\circ$ – оператор поэлементного умножения,

$r \in R^m$ – вектор-маска, состоящий из бернуллиевских случайных величин, которые равны 1 с вероятностью $p = 0.5$.

При обучении в алгоритме обратного распространения ошибки мы учитываем только «незамаскированные» связи, т.е. рассматриваем векторы c' и w' (в соответствии с формулами (4) и (5)).

Во время тестирования сети, мы умножаем обученные веса на *p*, и используем полученные значения весов. Т.е. на этапе применения модели каждая координата вектора-маски *r* равна *p*.

Стоит отметить, что дополнительно можно также использовать l2-регуляризацию для координат центра разброса значений признаков, т.е. весов синаптических связей на последнем слое.

Как было сказано ранее, последний слой должен выполнять объединение всех признаков в единый вектор и выявление на основании их значений аномальных данных. При этом в существующих работах [5,6] предлагается использовать несколько нейронов на выходном слое. Таким образом, в существующих работах, связанных с применением сверточных нейронных сетей для анализа текстовых данных, предлагается лишь минимизация целевых функций с целью решения задач многоклассовой классификации.

Поскольку реализуемая нейронная сеть должна решать задачу одноклассовой классификации, в предложенном решении последний слой имеет всего один выход. При этом мы используем гипотезу компактности, предполагая, что все вектора признаков для обучающей выборки должны лежать близко друг к другу. В связи с этим предположением мы можем построить выходной слой таким образом, чтобы полученный результат непосредственно зависел от расстояния, построенного сверточной частью вектора признаков до некоторого центра распределения признаков, найденного в процессе обучения. На основании этого, в реализуемом решении в качестве функции активации на последнем слое предлагается использовать радиально-базисную функцию, описываемую формулой (6).

$$f(x) = e^{-(c'-w')^2}, \quad (6)$$

где c' – значение признаков, полученное после применения операции отсечения и описываемое формулой (4),

w' – координаты центра распределения сверточных весов и описываемые формулой (5).

В качестве целевой функции используется функция, описываемая формулой (7).

$$g(x) = -\log(f(x)), \quad (7)$$

где $f(x)$ – значение функции активации, описываемой формулой (6).

Подставив явное выражение для $f(x)$ и осуществив ряд преобразований, мы придем к формуле (8).

$$g(x) = (x'-w')^2, \quad (8)$$

Таким образом, в процессе обучения посредством минимизации функции ошибки мы ищем центр распределения сверточных признаков с возможной l2-регуляризацией. При этом мы стараемся приблизить значения признаков на тренировочной выборке к искомому центру.

3. Экспериментальная оценка предлагаемого решения

С целью проведения экспериментальной оценки предлагаемого решения предлагается произвести сравнение данного метода с рядом классических методов, таких как одноклассовый метод опорных векторов, репликаторные нейронные сети, а также метод, основанный на ортонормированной неотрицательной матричной факторизации матрицы весов признаков. Как правило, для экспериментальной оценки эффективности работы того или иного алгоритма используются различные интегральные метрики качества. Одна из самых широко используемых метрик – это площадь под ROC-кривой (AUC-ROC). Анализ существующих работ по тематикам, схожим с тематикой настоящей работы, показывает, что, как правило, для оценки качества алгоритма выявления аномалий при работе пользователя применяется именно эта метрика ([4]). Поэтому она и будет использована в настоящей работе для оценки эффективности предлагаемого алгоритма.

3.1 Набор данных

В целях проведения наиболее качественного сравнения работы алгоритмов классификации, было решено производить сравнение на одном и том же наборе данных. В качестве исследуемого набора данных было выбрано множество данных, полученных от работников компании ENRON в 2000 и 2001 году. В данной работе были рассмотрены электронные письма, а также текстовые документы, приложенные к данным письмам. При этом, под текстовыми документами мы будем понимать файлы текстовых форматов (DOC, RTF, PDF). Для достижения репрезентативности экспериментальной выборки, были отобраны лишь те пользователи, суммарный объем всех текстовых файлов по которым превосходит 1 Гб. Таким образом, было отобрано 11941 текстовых документов для 15 пользователей. Данные для каждого пользователя были разделены на экспериментальные периоды в 6 недель с шагом в 2 недели. Каждый экспериментальный период был разделен на 4 недели для построения модели и 2 недели – для ее применения. Более подробное описание данного набора приведено в [7].

3.2 Качественное сравнение предлагаемого метода с классическими алгоритмами

В рамках данной работы были произведены эксперименты по построению модели с использованием одноклассового метода опорных векторов (SVM). При этом наилучшие результаты были получены с

помощью бинарного представления весов термов (т.е. 0, если терм не встречается в документе и 1 – иначе), а также представления весов с использованием метрики tf-idf (см., например, [4]). При этом было отмечено, что наиболее точных результатов можно добиться путем выделения из словаря среди всех термов тех, для которых среднее значение веса по всем документам в выборке будет наибольшим. В ходе экспериментов было произведено исследование зависимости качества классификации от размера словаря. Также в работе [7] приводится экспериментальная оценка метода, основанного на неотрицательной ортонормированной факторизации матрицы весов термов. Данные эксперименты также проводились на основании указанного выше набора данных. Более того, проводились эксперименты, связанные с применимостью репликаторных нейронных сетей в рамках данной задачи. При этом, целесообразным было проанализировать различные реализации с использованием разного количества скрытых слоев.

Говоря о предлагаемом решении, необходимо рассмотреть различные значения следующих параметров: количество используемых фильтров, число эпох обучения, значение коэффициента l2-регуляризации. В ходе серии экспериментов было выявлено, что наиболее оптимальным является 400 эпох обучения. После выбора числа эпох обучения была проведена серия экспериментов по выбору значения коэффициента l2-регуляризации, а также по подбору числа фильтров определенного размера (на основании работ [5], [6] использовалось по n фильтров размером 3, 4 и 5, где $n \in N$ – некоторое фиксированное число, которое подбиралось экспериментальным путем. Таким образом, в документе будет всего $n * 3$ признаков).

Результаты всех проведенных экспериментов вынесены в таблицу 1.

Метод классификации	**Медиана**
Опорных векторов (bin, 30 слов)	0.8862
Опорных векторов (bin, 50 слов)	*0.8921*
Опорных векторов (tf-idf, 30 слов)	0.89
Опорных векторов (tf-idf, 50 слов)	0.8783

ОНМФ (евклидова норма)	0.8996
ОНМФ (максимум)	*0.9065*
RNN (один скрытый слой)	*0.7576*
RNN (три скрытых слоя)	0.7298
Предложенный метод (l2_coef=0.5, n_filter=70)	0.8088
Предложенный метод (l2_coef=100, n_filter=70)	0.8801
Предложенный метод (l2_coef=10000, n_filter=50)	0.9
Предложенный метод (l2_coef=10000, n_filter=70)	***0.9207***
Предложенный метод (l2_coef=10000, n_filter=128)	0.88

Таблица 1. *Результаты проведенных экспериментов. bin – бинарное представление весов, tf-idf – представление весов с использованием метрики tf-idf, l2_coef – коэффициент l2-регуляризации, n_filter – число фильтров фиксированного размера*

Таким образом, стоит отметить, что наилучший результат можно наблюдать в случае предложенного метода с использованием коэффициента l2-регуляризации, равным 10000, и при 70 фильтрах, соответствующих размерам окна 3, 4 и 5.

Заключение

В рамках данной работы были рассмотрены методы обнаружения аномалий в работе пользователя с текстовыми данными. Предложен новый алгоритм обнаружения аномалий, решающий задачу одноклассовой классификации с использованием сверточных нейронных сетей. Также была проведена экспериментальная оценка построенного метода, показавшая, что данное решение показывает лучший результат по сравнению с рассмотренными классическими методами решения задачи обнаружения аномалий.

Литература

1. Chandola V., Banerjee A., Kumar V. Anomaly Detection: A Survey // ACM Computing Surveys. 2009. 41. № 3. P. 15:1-15.58.

2. K. Ramakrishnan, W. Hyenkyun, P. Haesun и A. Charu C. Outlier Detection for Text Data: An Extended Version. [PDF] (https://arxiv.org/abs/1701.01325v1)

3. Королёв В. Ю., Корчагин А.Ю., Машечкин И. В., Петровский М. И., Царев Д. В. Применение временных рядов в задаче фоновой идентификации пользователей на основе анализа их работы с текстовыми данными // Труды Института системного программирования РАН. 2015. 27. № 1. С. 151-172.

4. Manevitz L. M., Yousef M. One-Class SVMs for Document Classification // The Journal of Machine Learning Research. 2002. № 2. P. 139-154.

5. Kim Y. Convolutional Neural Networks for Sentence Classification [PDF] (https://arxiv.org/pdf/1408.5882v2.pdf).

6. Britz D. Implementing a CNN for text classification in Tensorflow (http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/).

7. Машечкин И. В., Петровский М. И., Царёв Д. В. Методы машинного обучения для анализа поведения пользователей при работе с текстовыми данными в задачах информационной безопасности // Вестник Московского университета. Серия 15: Вычислительная математика и кибернетика. 2016. № 4. С. 33-39.

8. Hawkins S., He H., Williams G., Baxter R. Outlier Detection Using Replicator Neural Networks // Data Warehousing and Knowledge Discovery. 2002. № 2454. P. 170-180.

9. В. Ю. Гудков, Е. Ф. Гудкова N -граммы в лингвистике // Вестник Челябинского государственного университета, т. № 24(239), pp. 69-71, 2011.

Закляков Р.Д., Петровский М. И., Попов И.С.

Разработка гибридных методов распознавания пользователя по особенностям его работы со стандартными устройствами ввода компьютера

Введение

В условиях современного информационного общества актуальной является задача аутентификации компьютерных пользователей. Классическими решениями данной задачи являются процедуры, основанные на анализе факта обладания некоторым уникальным знанием, например, парой логин/пароль, и процедуры, основанные на анализе факта обладания некоторым физическим предметом или физиологической особенностью, например, ключ-картой или специфическим отпечатком пальцев. Вышеупомянутые типы аутентификации обладают рядом недостатков, среди которых: уникальные знания могут быть утеряны, переданы третьим лицам или скомпрометированы (например, большинство пользователей используют одинаковую пару логин/пароль для различных ресурсов), а для анализа факта обладания некоторым физическим предметом или физиологической особенностью зачастую требуется дополнительное оборудование, причем безопасность таких систем может быть скомпрометирована подменой ключа аутентификации (например, может быть изготовлена силиконовая копия отпечатка пальца, имеющего необходимый рисунок).

Альтернативным решением задачи аутентификации являются процедуры анализа характера взаимодействия пользователя со стандартными устройствами ввода компьютера (например, анализ скорости ввода с помощью клавиатуры и анализ скорости перемещения курсора). Основными преимуществами данного подхода являются низкая стоимость реализации и внедрения в существующие компьютерные системы, а также высокая устойчивость системы аутентификации к подмене – результаты экспериментов говорят о том, что подделать характер ввода человека чрезвычайно сложно даже в том случае, если атакующий имел возможность наблюдать за тем, как работал легитимный пользователь [1].

Исторически сложились два основных направления в области методов анализа легитимности пользователей по характеру их взаимодействия с манипуляторами: анализ поведения пользователя в ходе его повседневной работы (фоновые методы) и анализ пользователя

при прохождении специальных сценариев, требующих от него произвести заранее предопределенную последовательность действий (статические методы). На практике, методы фоновой аутентификации имеют более низкую точность по сравнению со статическими методами, что может быть объяснено отсутствием непосредственной привязки к данным, с которыми работает пользователь, большой размерностью пространства признаков по сравнению с обучающей выборкой, большим количество "шумовых" действий пользователя, не несущих никакой информативности (например, хаотичные движения мышки, когда пользователь о чем-то задумался).

В статье рассматривается задача разработки технологии на основе применения статических методов распознавания пользователей по особенностям их взаимодействия со стандартными манипуляторами компьютера.

Решение рассматриваемой задачи предполагает проведение следующих этапов: разработки специальных сценариев работы пользователя, вынуждающих его произвести заранее предопределенную последовательность действий, этапа исследования возможности построения обучающего набора данных для статической модели на основе данных фонового сбора, этапа выбора оптимальных моделей представления данных для отображения характера ввода пользователя в векторном виде, а так же итоговой оценки качества разработанных сценариев и выбранных методов представления данных.

1. Предлагаемые сценарии работы

Для рассматриваемой задачи в рамках данной работы предлагается использовать следующие сценарии работы, побуждающие пользователя произвести заранее предопределенную последовательность действий:

1. клавиатурный сценарий «Случайная последовательность символов»: пользователю предлагается ввести некоторую короткую случайно сгенерированную последовательность символов. Визуально соответствует классическому форм-фактору CAPTCHA;
2. клавиатурный сценарий «Фиксированная последовательность символов»: пользователю предлагается ввести определенное слово из некоторого словаря слов. Визуально соответствует форм-фактору CAPTCHA, в котором в качестве фраз используются полноценные слова;
3. сценарий по мышке «Графическая подпись»: пользователю предлагается последовательно соединить курсором набор точек. Этот набор точек может представлять из себя как

некоторую картинку (например, прописную букву А), так и некоторый упорядоченный набор точек, равномерно расположенных по всему экрану (аналогично расположению точек на экране разблокировки Android-устройств, см. Рисунок 1);

4. сценарий по мышке «Графическая CAPTCHA»: пользователю предлагается выбрать из небольшого набора картинок те из них, которые соответствуют некоторому критерию (пример данного сценария см. Рисунок 2). Визуально соответствует классическому форм-фактору CAPTCHA с помощью картинок.

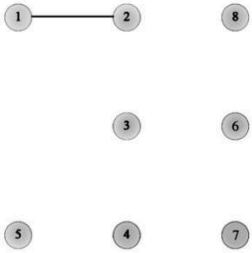

Рисунок 1. *Пример сценария типа "Графическая подпись" в вариации "Пароль Android-устройств"*

Рисунок 2. *Пример сценария типа "Графическая CAPTHA"*

2. Построение обучающей выборки

По результатам предварительных экспериментов был сделан вывод о том, что использование фоновых моделей для распознавания характера прохождения вышеперечисленных сценариев не дает необходимой точности аутентификации, поэтому необходимо для каждого экземпляра сценария строить отдельную модель пользователя (т.е., для сценариев типа «Фиксированное слово» будут построены отдельные модели по каждому из слов, которые будет предлагаться ввести пользователю). Ниже представлены методы получения обучающей выборки для обучения моделей, рассматриваемые в данной работе:

2.1. Статический метод построения обучающей выборки

Данный метод предполагает, что первые *n* отображений специальных сценариев работы предназначены только для получения обучающих данных и построения модели.

Основной минус данного подхода – не используются данные фонового сбора, хотя пользователь мог уже довольно долго работать за машиной, и в его действиях уже можно было выделить некоторые характерные черты.

2.2. Метод построения обучающей выборки на основе данных фонового сбора

Данный метод предполагает, что начальные модели распознавания по характеру прохождения специальных сценариев работы будут строиться на основе данных фонового сбора (заранее заметим, что такие изначальные модели будут несколько более низкого качества, чем если бы мы использовали модели, обученные полностью на данных, полученных только при прохождении специальных сценариев, однако, мы можем постепенно удалять из обучающей выборки вводы из фонового сбора по мере того, как пользователь будет проходить данные сценарии работы все больше и больше раз, и начиная с некоторого момента, модели будут обучены только на чистых данных, полученных в ходе статического сбора).

Алгоритм получения обучающей выборки следующий:

1. поток событий проходит процедуру очистки от "потерянных" событий:
    a. последовательно "сверху вниз" удаляются все события типа "отпускание клавиши", для которых не было нажатия;

b. последовательно "снизу вверх" удаляются все события типа "нажатие клавиши", для которых не было отпускания;

2. выделяются события-маркеры, соответствующие клавишам-разделителям (*пробел*, *Tab*, *End*, *Home*...) и временным паузам в ходе ввода;
3. выделяются наборы событий, лежащих между найденными на предыдущем шаге событиями-маркерами (таким образом получаем события, относящиеся к вводу одного слова);
4. по каждому слову строим статистику по количеству его вводов (опционально возможно строить статистику не только по всему слову, но и по его префиксу, например, можно упрощенно считать, что характер ввода префикса слова "спортивный" не сильно отличается от характера ввода слова "спорт", поэтому в список слов и соответствующих им событий будут добавлены не только сами слова, но и их префиксы);
5. для тех слов, для которых зафиксировано достаточное количество вводов, строится обучающая выборка из событий, соответствующих вводу этого слова, и регистрируется новая провокация.

3. Описание модели представления данных

После того, как была подготовлена обучающая выборка, необходимо отобразить ее в набор численных характеристик. Выделение признаков, на основе которых в дальнейшем будет формироваться вектора признаков, является одним из самых важных этапов построения решения к рассматриваемой задаче.

3.1. Признаковое пространство для клавиатурных данных

В области выделения признаковых пространств для описания характера взаимодействия пользователя с **клавиатурой** существуют следующие два основных подхода [1, 2]:

1. **подход на основе анализа одиночных нажатий**: Данный метод представления данных является наиболее часто используемым методом среди работ, посвященных распознаванию пользователей на основе характера их работы с клавиатурой. Двумя основными рассматриваемыми величинами при данном подходе являются длительность зажатия клавиши (*hold*) и задержка между двумя последовательными нажатиями (*latency*). По каждой клавише порождается отдельная пара признаков;
2. **подход на основе анализа *n*-графов нажатий**: В задачах, связанных с распознаванием на основе клавиатурного

почерка, под *n*-графами подразумеваются характеристики, которые можно извлечь из пользовательского ввода, если рассматривать его не как набор независимых одиночных нажатий, а последовательностями по n нажатий. Таким образом, нажатия рассматриваются не как изолированные события, а как совокупность событий в некоторой окрестности, что потенциально может повысить качество выходного вектора применительно к задаче распознавания по клавиатурному почерку. Каждый *n*-граф порождает отдельную группу признаков. Чаще всего используемая $n = 2$. Такие *n*-графы называются диграфами и имеют простую интерпретацию, привязанную к окружающему миру: во многих языках есть устоявшиеся двухбуквенные сочетания, чаще других встречающиеся в словах, например, в английском языке диграфами могут служить комбинации *th*, *ch*, *sh*, а в русском – *mь*, *пр*, *ис*.

Характеристиками диграфов могут служить следующие величины (их графическая интерпретация представлена на Рисунке 3):

a. общее время диграфа (*du*, $t_2^{up} - t_1^{down}$);
b. временем между нажатиями первой и второй клавиши (*dd*, $t_2^{down} - t_1^{down}$);
c. временем между отпусканием первой и нажатием второй клавиши (*ud*, $t_2^{down} - t_1^{up}$, то же самое, что и *latency*);
d. временем между отпусканием первой и отпусканием второй клавиши (*uu*, $t_2^{up} - t_1^{up}$).

Где t_i – время взаимодействия с *i*-ой клавишей диграфа, *down* соответствует нажатию соответствующей клавиши, а *up* – отпусканию.

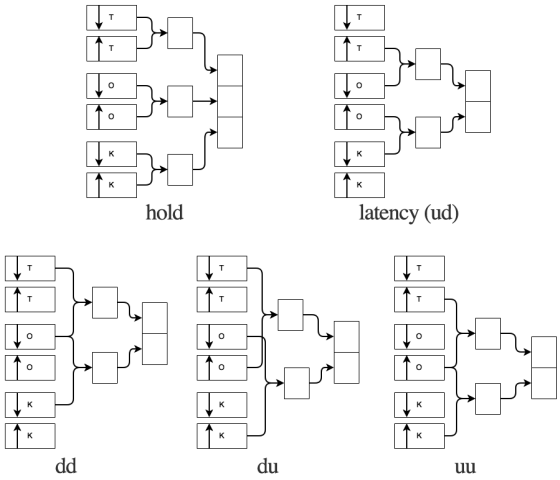

Рисунок 2. *Графическая интерпретация клавиатурных признаков*

На основании предварительных экспериментов были выбраны признаки "Длительность нажатия", "Длительность между нажатиями", "Длительность между последовательными нажатиями двух клавиш", "Длительность между последовательными отпусканиями двух клавиш", так как они показали наилучшие результаты.

3.2. Признаковое пространство для данных, полученных с помощью манипулятора «мышь»

На основе анализа работ [3-5] было выделено следующее признаковое пространство для данных, собранных с помощью мышки: сначала все записанные координаты курсора нормируются путем деления координат на ширину и высоту экрана. Далее, вычисляются три простых характеристики:

1. средняя скорость перемещения курсора;
2. средняя скорость по каждому из направлений перемещения;
3. максимальная скорость по каждому из направлений перемещения.

На основе предварительных экспериментов было установлено, что оптимальное число направлений равняется девяти. Графическую интерпретацию деления направлений на группы см. Рисунок 4.

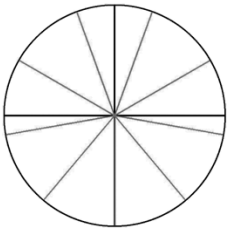

Рисунок 3. *Графическая интерпретация деления направления перемещений на группы*

После вычисления простых признаков вычисляются два комплексных признака, характеризующих особенности того, как пользователь перемещает курсор между двумя последовательными точками: средняя глубина траектории и распределение модуля среднего ускорения в зависимости от части траектории. Рассмотрим способ вычисления этих признаков подробнее.

Средняя глубина траектории между двумя точками описывает, насколько сильно пользователь отклоняется от прямой, соединяющей две конечные точки траектории. Так как определение точки середины траектории по координатам курсора, собранным через равные промежутки времени, является довольно сложной вычислительной задачей (см. верхнюю часть Рисунка 5), данный признак вычисляется по модифицированному правилу: для каждой тройки последовательных координат траектории вычисляется расстояние между прямой, соединяющей первую и третью точку, до второй точки, после чего все вычисленные глубину усредняются. Графическую интерпретацию данного метода см. нижнюю часть Рисунка 5.

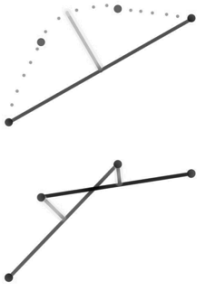

Рисунок 4. *Графическая интерпретация метода определения средней глубины траектории*

Распределение модуля среднего ускорения в зависимости от части траектории характеризует то, насколько быстро пользователь ведет курсор в начале траектории, ее середине и ее конце. Для этого вся

траектория делится на три равных части, и для каждой из частей вычисляется средняя скорость перемещения курсора.

4. Используемые методы классификации

Если этап выделения признаков отвечает за отображение особенностей ввода пользователя в точки в некотором многомерном пространстве, то этап бинарной классификации в задаче распознавания пользователя по его особенностям ввода в условиях неограниченного множества людей, имеющих потенциальный доступ к машине, отвечает непосредственно за то, чтобы по набору точек, вычисленных по данным легитимного пользователя, ответить на вопрос: принадлежит ли новая точка, поступившая на вход, классу точек легитимного пользователя.

В качестве рассматриваемых классификаторов были выбраны одноклассовый метод опорных векторов [6] (SVM, наиболее популярный метод классификации в задачах распознавания по характеру взаимодействия со стандартными устройствами ввода [1, 7]) и его модификация, Нечеткий метод поиска исключений на основе потенциальных функций [8], показывающая на ряде задач более высокое качество распознавания [8, 9].

Параметры SVM предлагается подбирать экспериментально. Для оценки отношения выбросовых векторов к общему числу векторов в обучающей выборке, необходимого для корректной работы Нечеткого метода поиска исключений на основе потенциальных функций, предлагается использовать представленный ниже метод, разработанный в рамках данной работы.

Работа алгоритма основывается на том предположении, что исключение одного вектора из большой обучающей выборки не должно существенно повлиять на точность модели. В дальнейшем в данной главе под термином «классификатор» мы будем подразумевать обученный алгоритм Нечеткого метода поиска исключений на основе потенциальных функций.

Формально, вычисляются две величины: *entropy suggested* ("кластер стал достаточно компактным после удаления очередного самого далекого вектора") и *empiric suggested* ("при удалении очередного самого удаленного вектора структура кластера стала меняться в негативную для точности модели сторону").

Entropy suggested вычисляется следующим способом: для каждого k от 0 до 0.6 с шагом 0.01 последовательно строятся классификаторы с параметром k и определяется величина энтропии внутри кластера. Как только величина энтропии внутри класса становится меньше 0.15, алгоритм заканчивает свою работу.

Empiric suggested вычисляется следующим способом: для каждого *k* от 0.01 до 0.61 с шагом 0.1 последовательно выполняются следующие шаги:

1. последовательно строятся *n* классификаторов с параметром *k*, обученных на подвыборках обучающего набора: $X_i = \{x_i \in X \mid i \neq n\}$;
2. далее, с помощью каждого классификатора вычисляется расстояние от исключенного вектора до построенного нечеткого кластера;
3. все расстояния агрегируются в один массив и сортируются по убыванию;
4. далее берется расстояние, лежащее в ячейке $\text{int}(k \times len(array))$, обозначаемое как $threshold_k$;
5. строится общий классификатор на всей обучающей выборке с параметром *k*, и для него определяется параметр энтропии $entropy_k$;
6. вычисляется значение $$j_k = \sqrt{(entropy_k - entropy_{k-1})^2 + (threshold_k - threshold_{k-1})^2}$$

Проход по *k* завершается, как только $j_k > j_{k-1}$.

5. Эксперименты

В рамках работы был произведен сбор тестовых данных. В сборе приняли участие 20 человек, каждый из которых прошел через 30+ итераций по каждому из типов специальных сценариев. Обучение пользовательских моделей для клавиатурного сценария типа «фиксированное слово» осуществлялся на основе данных фонового сбора, осуществлявшегося на протяжении 8+ часов.

Для оценки качества полученных моделей использовалось значение площади под ROC-кривой (ROC-AUC). ROC-AUC является инвариантной относительно отношения количества векторов разных классов в тестовой выборке и обозначает вероятность того, что классификатор присвоит больший вес положительному примеру, чем отрицательному. Чем ближе значение к 1, тем более качественной считается модель (см. таблица 1).

Тип сценария	**SVM, ROC-AUC**	**Нечеткий метод, ROC-AUC**
«Случайная последовательность символов»	0.571	0.563
«Фиксированное слово»	0.862	0.895
«Графическая подпись» в вариации полноценной картинки	0.587	0.578
«Графическая подпись» в вариации «экран блокировки Android-устройств»	0.689	0.695
«Графическая CAPTCHA»	0.595	0.565

Таблица 1. *Результаты экспериментов*

Для сравнения приведена средняя точность [2, 3] для фоновых и статических моделей для каждого из манипуляторов (см. Таблицу 2).

	Фоновая аутентификация	**Статическая аутентификация**
Клавиатура	0.87	0.94
Мышка	0.77	0.78

Таблица 2. *Средняя точность фоновых и статических моделей*

Как видно из вышеприведённых результатов, только клавиатурный сценарий типа «Фиксированное слово» смог превзойти точность базовых фоновых моделей. Однако, в случае, когда системе необходимо в короткие сроки произвести аутентификацию пользователя с большой степенью уверенности, в комбинации с «Фиксированным словом» можно использовать и сценарий типа «Графическая подпись» в вариации «экран блокировки Android-устройств». Хоть точность последнего и несколько ниже точности фоновых моделей по мышке, ему не требуется набора событий в течение некоторого времени для наполнения окна событий, и поэтому аутентификация может быть произведена в очень короткие сроки.

Заключение

В рамках данной работы была рассмотрена задача разработки специальных сценариев работы, которые можно интегрировать в повседневный цикл взаимодействия пользователя с компьютером, побуждающих его произвести заранее предопределенную последовательность действий, а так же задача анализа характера выполнения данных последовательностей.

Были разработаны ряд сценариев, которые могут быть встроены в систему для осуществления сбора данных, попадающих под понятие статической аутентификации. Основными преимуществами данных сценариев является то, что эти сценарии вписываются в повседневный цикл взаимодействия пользователя с компьютером, не повлияют на его манеру взаимодействия с системой и не вызовут негативной реакции из-за необходимости производить лишние действия для прохождения аутентификации. Потенциальный злоумышленник может даже не подозревать, что система аутентификации проводит проверку его легитимности, что так же положительным образом сказывается на итоговый уровень защищенности системы.

На основе предварительных экспериментов были определены оптимальные признаковые пространства для клавиатуры и мышки, а так же предложен метод построения статических клавиатурных моделей на основе данных фонового сбора. Предложен метод оценки отношения выбросовых векторов в обучающей выборке к их общему числу. Для экспериментальной оценки качества предложенных подходов были проведены тестовые запуски с применением различных классификаторов (SVM и Нечеткого метода поиска исключений на основе потенциальных функций).

Экспериментально была показана применимость клавиатурного сценария типа «Фиксированное слово», модель которого обучена на данных фонового сбора, а так же показана потенциальная пригодность сценария типа «Графическая подпись» в вариации «экран блокировки Android-устройств» для повышения общей надежности фоновых систем аутентификации, анализирующих характер взаимодействия пользователя со стандартными устройствами ввода компьютера.

Литература

1. Teh P. S., Teoh A. B. J., Yue S. A survey of keystroke dynamics biometrics //The Scientific World Journal. – 2013. – Т. 2013.
2. Banerjee S. P., Woodard D. L. Biometric authentication and identification using keystroke dynamics: A survey //Journal of Pattern Recognition Research. – 2012. – Т. 7. – №. 1. – С. 116-139.
3. Bailey K. O., Okolica J. S., Peterson G. L. User identification and authentication using multi-modal behavioral biometrics //Computers & Security. – 2014. – Т. 43. – С. 77-89.
4. Shen C. et al. Performance evaluation of anomaly-detection algorithms for mouse dynamics //Computers & Security. – 2014. – Т. 45. – С. 156-171.
5. Pao H. K. et al. Trajectory analysis for user verification and recognition //Knowledge-Based Systems. – 2012. – Т. 34. – С. 81-90.
6. Roemer V. Introduction to One-class Support Vector Machines [Электронный ресурс]. - Режим доступа: \url{http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/} (дата обращения: 20.05.2017)
7. Bailey K. O., Okolica J. S., Peterson G. L. User identification and authentication using multi-modal behavioral biometrics //Computers & Security. – 2014. – Т. 43. – С. 77-89.
8. Kazachuk M. et al. One-Class Models for Continuous Authentication Based on Keystroke Dynamics //International Conference on Intelligent Data Engineering and Automated Learning. – Springer International Publishing, 2016. – С. 416-425.
9. Petrovskiy M. A fuzzy kernel-based method for real-time network intrusion detection //International Workshop on Innovative Internet Community Systems. – Springer Berlin Heidelberg, 2003. – С. 189-200.

Красняков Е.И., Попов И.С.

Аутентификация пользователя мобильного устройства по характеристикам работы с виртуальной клавиатурой

Введение

Аутентификация пользователей в различных системах доступа является весьма актуальной темой в области информационных технологий. Что касается смартфонов, то в современном мире люди всё чаще и чаще их используют для хранения там конфиденциальной информации, а также для доступа к онлайн-сервисам, вмешательство злоумышленника в которые может привести к весьма плачевным последствиям. Существующие методы аутентификации смартфонов (например, по цифровому или графическому паролю) недостаточно эффективны, поскольку эту информацию нетрудно перехватить или вычислить. Что касается биометрических естественных сканнеров, то во-первых, ими оборудованы далеко не все устройства, во-вторых, они требуют определённых материальных затрат, и наконец в-третьих, ими не всегда можно успешно воспользоваться (например, сканнер отпечатков пальцев может заляпаться и не работать с влажными руками). Одно из наиболее перспективных направлений защиты подобных систем от несанкционированных воздействий – реализация методов аутентификации пользователей на основе поведенческой биометрии.

Идеальная биометрическая характеристика человека (БХЧ) обычно обладает универсальностью, уникальностью, стабильностью и собираемостью. Универсальность означает наличие биометрической характеристики у каждого человека. Уникальность означает, что не может быть двух людей, имеющих идентичные значения БХЧ. Стабильность — независимость БХЧ от времени. Собираемость — возможность получения биометрической характеристики от каждого человека.

Биометрические характеристики можно разбить на два основных класса [1]:

1) Естественные (отпечаток пальца, форма лица, сетчатка глаза, голос и т. д.). Системы распознавания, использующие эти данные, достаточно эффективно справляются со своей задачей, однако чаще всего для реализации такой системы требуется дорогостоящее оборудование и специфическое ПО;

2) Поведенческие (клавиатурный почерк, манера управления курсором мыши, работа с джойстиком и др.), которые собирают информацию о поведении человека при использовании какого-либо технического устройства. Для анализа таких характеристик не требуется специального технического оборудования, и их несложно совместить с другим ПО в информационных системах.

В этой работе будет рассматривается именно второй класс биометрических характеристик, а точнее - аутентификация пользователей по их манере ввода символов с экранной клавиатуры. Преимуществами такого метода аутентификации являются простота реализации и внедрения; отсутствие требований от пользователя каких-либо дополнительных действий, кроме привычных; а также возможность скрытой аутентификации.

1. Задача аутентификации пользователей

В текущей работе рассматривается задача аутентификации пользователей мобильных устройств по характеристикам работы с экранной клавиатурой. Для выполнения этой задачи были сформулированы следующие этапы.

1.1. Этапы решения

- Провести исследование наиболее распространённых алгоритмов, используемых в биометрических системах статической аутентификации (см. 1.2), основанных на клавиатурном почерке (в частности, при вводе текста с клавиатуры смартфона);
- Разработать мобильное приложение для сбора биометрических данных пользователей относительно ввода ими заданного текста с экранной клавиатуры устройства;
- Реализовать ПО, преобразующее собранные данные в вектора признаков в соответствии с исследованными моделями;
- Произвести экспериментальный анализ качества классификации выбранных алгоритмов и качества выбранных моделей, применённых к собранным пользовательским данным;
- Реализовать модуль аутентификации iOS-устройства, используя в нём алгоритмы и модели, показавшие лучшие результаты в проведённых ранее экспериментах.

1.2. Биометрическая аутентификации пользователя

Под аутентификацией подразумевается процедура проверки подлинности пользователя, которая позволяет удостовериться в том, что человек, предоставивший некий идентификатор, на самом деле является именно тем человеком, идентификатор которого он использует.

Аутентификация делится на два класса – статическая и динамическая. В первом случае она проводится на некотором фиксированном паттерне, а для обучения системы пользователю нужно многократно ввести одни и те же данные. Динамическая же аутентификация происходит на произвольном наборе данных, а анализ действий пользователя происходит в фоновом режиме.

Главный принцип биометрической аутентификации на основе поведения пользователя схож во многих системах, которые на этом специализируются [2]:

1) В начале работы происходит многократная запись биометрических образцов пользователя.

2) Далее из этих образцов производится выделение необходимых уникальных характеристик для построения биометрической модели пользователя.

3) После этого каждый новый биометрический образец сравнивается с построенной моделью, на основе чего выясняется, принадлежит ли полученный образец владельцу устройства.

1.3. Требования к аутентификации

Относительно аутентификации обычно выделяются следующие значимые требования [6, 9]:

1) Небольшое количество входных данных: все данные для входа должны быть невелики. В этой работе максимум для длины парольной фразы будет установлен в 20 символов. Это позволит ввести её достаточно быстро (в среднем, менее чем за полминуты), а значит и упростить процессы обучения и тестирования. То есть каждый пользователь сможет в любой удобный момент быстро дообучить классификатор или протестировать его.

2) Полезность входных данных: из ввода того или иного набора символов можно извлечь достаточное количество признаков для успешного обучения и тестирования.

3) Высокая скорость работы: обучение классификатора, а также сам процесс пользовательской аутентификации должны происходить за малый промежуток времени. Для небольших данных, как в этой работе, на обучение обычно требуется пара десятков вводов. Сопоставляя со средним временем ввода в первом пункте, обучение в среднем займёт 10 минут, а каждый сеанс аутентификации – полминуты.

4) Высокая точность распознавания: система должна достаточно точно отличать человека, на котором происходило обучение, от других людей. Вероятность допуска злоумышленника в подобных системах рекомендуется делать не выше 1-2%, а общую вероятность верного распознавания - не ниже 90%.

1.4. Критерии оценки результата

Для того чтобы понять, насколько пригоден тот или иной подход, воспользуемся следующими оценками [3, 11]:

1) *FFR (False Positive Rate)* – ошибка первого рода, заключающаяся в опровержении верной гипотезы.

2) *FAR (False Acceptance Rate)* - ошибка второго рода, заключающаяся в принятии ложной гипотезы.

3) *ROC-кривая* – это график, позволяющий оценить качество бинарной классификации. Он отображает соотношение между долей объектов от общего количества носителей признака, верно классифицированных, и долей объектов от общего количества объектов, не несущих признака, ошибочно классифицированных [6].

4) *AUC* — площадь, ограниченная ROC-кривой и осью доли ложных положительных классификаций [6].

5) *Общая точность* – отношение правильно данных классификатором ответов к общему количеству тестов.

2. Используемые методы для построения модели

2.1. Метод представления данных

Важным пунктом для решения поставленной задачи является удобное представление данных для классификатора. Чтобы эффективно использовать существующие алгоритмы классификации, необходимо использовать модель, отображающую в том или ином формате действия пользователя в виде числового вектора. Этот вектор создаётся как последовательность определённого набора признаков, собираемых с того или иного устройства ввода (здесь устройство ввода – это экранная клавиатура).

Для удобства классификации каждый вектор будет нормализован. Под нормализацией подразумевается преобразование заданного вектора в вектор того же направления, но с единичной длиной. В данной работе используется Евклидова нормировка, которая в большинстве случаев ощутимо улучшает качество классификации [3, 7].

При построении модели использовались характеристики, которые чаще всего используются для решения задач с клавиатурой, аналогичной представленной в этой работе. Это [4, 5]:

- время нажатия клавиши;
- время между нажатиями клавиш;
- диграфы и триграфы (время между нажатием первой клавиши и отпусканием второй/третьей);
- общая скорость ввода всей последовательности.

2.2. Алгоритмы классификации

В экспериментах будет использовано три различных классификатора: KNN [10], SVM [8] и нечёткий метод поиска исключений [12]. Этот выбор обусловлен успешностью их использования для задач подобного рода [4]. С помощью этих классификаторов и будет определяться, «свой» или «чужой» человек вводит ту или иную последовательность.

2.3. Метод принятия решений

Пожалуй, один из самых распространённых и легко реализуемых методов – это аутентификация по порогу. В результате обучения классификатора создаётся некоторый эталонный вектор признаков, по которому характеризуется конкретный пользователь. При попытке авторизоваться программа сравнивает вектор признаков человека, который хочет войти в систему, с «эталонным» пользователем, вычисляя определённым способом расстояние между этими векторами. Если оно меньше некоторого заранее предопределённого числа (порога), пользователь проходит авторизацию, в противном случае доступ к системе для него закрыт [7].

Этот метод, как самый универсальный, и будет использован для построения модуля аутентификации.

3. Исследование и построение решения

3.1. Этапы решения поставленной задачи

1) Сбор данных, отражающих динамику работы пользователя с экранной клавишной панелью.

2) Построение модели представления данных.

3) Выбор метода машинного обучения для аутентификации пользователя.

4) Выбор метода принятия решений, который на основе результатов алгоритма классификации (как упоминалось выше, исследуется работа SVM, KNN и нечёткого метода поиска исключений) выносит окончательное решение относительно аутентификации пользователя.

5) Построения iOS-модуля аутентификации, использующий для своей работы последовательности данных и классификаторы, показавших лучшие результаты в ходе экспериментов.

3.2. Сбор данных

Чтобы понять, какие данные и классификаторы необходимо использовать для реализации успешного модуля аутентификации внутри смартфона, требуется собрать с некоторой группы лиц эти данные и их проанализировать.

Для сбора биометрических данных пользователей мобильных устройств было разработано специальное приложение для iOS ОС, которое записывает нужную информацию в специальном формате в файл. Далее эти файлы пересылались в хранилище компьютера с помощью e-mail, после чего происходил их анализ.

В сборе данных каждой текстовой последовательности приняло участие от 6 до 10 человек (в зависимости от конкретной последовательности). С каждого человека было собрано, в среднем, по 15 вводов каждой строки. С учётом длин вводимых строк (от 15 до 27 символов) такое количество вводов вполне допустимо для обучения [13]. Для ввода предлагались следующие текстовые последовательности:

1) ‘life is beautiful';
2) 'have a nice day';
3) 'good luck to you';
4) 'strawberry banana and hello'.

3.3. Приложение для сбора данных

В процессе работы с этим приложением каждому пользователю предлагалось ввести определённое количество раз каждую из четырёх последовательностей слов, которые заранее были введены в память устройства. Последовательности вводятся по порядку. При каждом нажатии в файл записывается информация о времени нажатия и времени отпускания клавиши.

Если при вводе человек случайно отвлёкся или, например, ответил на вызов телефона, он может нажать кнопку "Отмена" во избежание некорректных данных.

Стоит также заметить, что поскольку Apple не позволяет фиксировать нужные нам характеристики через встроенную клавиатуру на iOS-устройствах, была нарисована собственная клавишная панель, которая максимально приближена к системной по внешнему виду.

После того, как пользователь ввёл нужное число последовательностей, сгенерированные файлы отправляются по почте с помощью кнопки «На почту».

На рис. 1 представлены несколько скриншотов работы этого мобильного приложения.

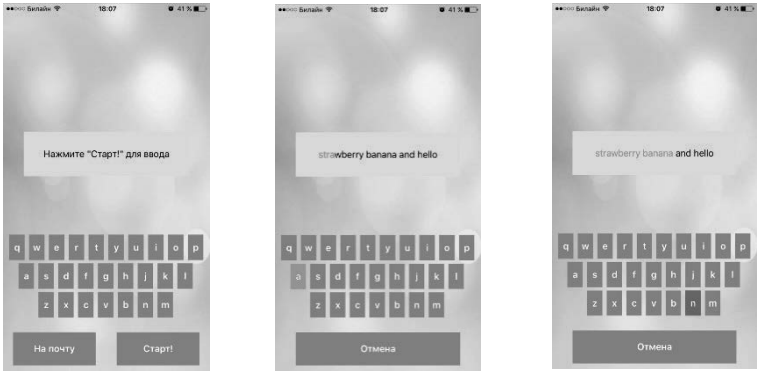

Рисунок 1. *Скриншоты с мобильного приложения по сбору*

3.4. Модель представления данных

Все данные в ходе экспериментов записывались в текстовый файл в формате: [КОД КЛАВИШИ] [СОБЫТИЕ] [ВРЕМЯ СОБЫТИЯ]. Этого представления было вполне достаточно для извлечения нужных характеристик в последующем анализе.

Время учитывается с помощью таймера. При обучении введённые параметры усредняются с помощью определённых алгоритмов, а при аутентификации устанавливается степень отклонения исходных данных с эталоном для данного пользователя с помощью заранее установленного порога.

3.5. Анализ качества работы классификаторов

Было принято решение исследовать работу классификаторов в среде Python 2.7 с использованием Open-Source библиотеки **Skicit-Learn**. Она предоставляет реализацию целого ряда алгоритмов для обучения с учителем (Supervised Learning) и обучения без учителя (Unsupervised Learning) через интерфейс для языка программирования Python. Scikit-learn построена поверх SciPy (Scientific Python), который должен быть установлен перед использованием scikit-learn. Данный стек включает в себя множество расширений, нам пригодится только NumPy - расширение Python, добавляющее поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых математических функций для операций с этими массивами. Использование Scikit-learn очень удобно в целях кластеризации, извлечения и отбора признаков и оценки эффективности алгоритмов.

Преимущество выбранного подхода для анализа заключается в простоте и качестве реализации выбранных выше методов классификации.

3.6. Реализация модуля аутентификации

Данный iOS-модуль реализовывался с помощью среды разработки XCode на языке программирования Objective C. Для классификации была задействована библиотека OpenCV, которая была адаптирована для выбранной среды разработки и использования её в языке Objective C.

В модуле аутентификации смартфона использовались реализации одноклассовых методов классификации, поскольку обучение будет производиться только на владельце устройства.

4. Эксперименты и полученные результаты

4.1. Подбор качественного набора данных

Ниже в таблицах 1-3 представлены результаты экспериментов по анализу качества классификации на выбранных четырёх последовательностей данных, а также ROC-кривые классификации. В соответствии с этими результатами следует обратить внимание, какие последовательности показывают наилучшую точность и какую модель аутентификации для них лучше использовать.

N посл-ти	Классификатор	Общая точность, %	FRR, %	FAR, %
1	**KNN ('Ball Tree')**	**86,0**	**59,4**	**3,0**
	SVM	81,7	68,7	6,0
	Fuzzy	82,3	68,7	5,3
2	**KNN ('Ball Tree')**	**82,7**	**67,7**	**4,8**
	SVM	81,4	67,7	6,4
	Fuzzy	74,4	67,7	15,2
3	**KNN ('Ball Tree')**	**85,5**	**45,2**	**6,6**
	SVM	78,3	67,7	9,9
	Fuzzy	84,9	64,5	2,5
4	**KNN ('Ball Tree')**	**86,1**	**55,2**	**3,5**
	SVM	75,7	69,0	13,0
	Fuzzy	77,8	69,0	10,4

Таблица 1. *Результаты классификации по экранной клавиатуре*

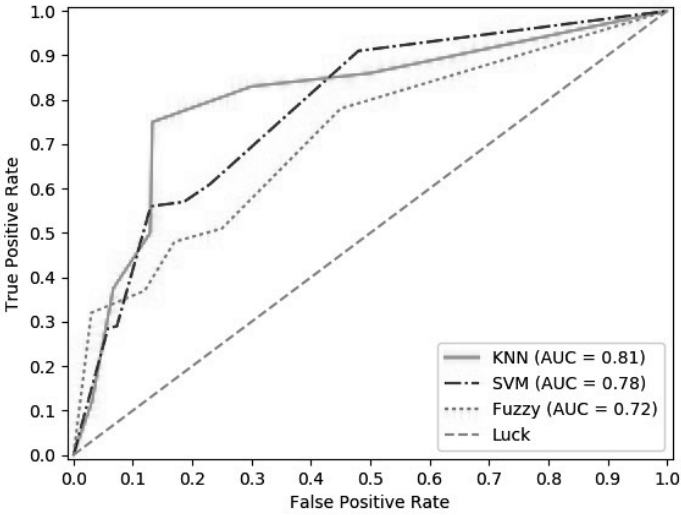

Рисунок 2. *ROC-кривые и AUC – Текст*

Как видно из табл. 1 и рис. 2, в той или иной степени неплохо себя показал каждый из классификаторов KNN, SVM и Fuzzy. Был достигнут весьма неплохой процент ошибок второго рода (3-5%). Высокий же процент ошибок первого рода объясняется относительно малой выборкой данных, которую удалось собрать с пользователей. Наибольшую же точность (86,1%) показали последовательности с номерами 1 и 2.

4.2. Реализация iOS-модуля аутентификации

Предыдущие эксперименты помогли выявить хорошо классифицируемые текстовые паттерны, теперь же необходимо провести эксперименты с ними в рамках алгоритмов классификации iOS-приложения. Проводить их разумней на данных, собранных с одного смартфона и посредством привычной для пользователей клавиатуры.

Предыдущие эксперименты обозначили также необходимость увеличения пользовательской выборки для снижения уровня ошибок первого рода.

Все эти факторы поспособствовали новому, расширенному сбору данных.

4.2.1. Расширенный сбор данных

1) До этого использовалась нарисованная вручную клавиатура, которая не являлась интуитивно привычной по сравнению с системной. Теперь же используется прототип встроенной iOS-клавиатуры, которую сложно отличить от оригинала (см. рис. 3)

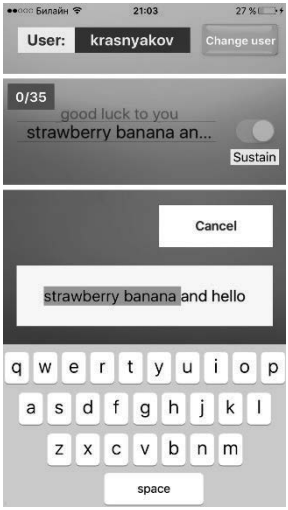

Рисунок 3. *Доработанная клавиатура для новых данных*

2) Выборка для нового сбора была расширена с 15 по 35 вводов каждой текстовой последовательности.

4.2.2. Результаты

Помимо общей точности, FRR и FAR была рассчитана ещё одна величина, обозначенная как «Evil_Probability». Она показывает вероятность проникновения злоумышленника, если он попытается войти в систему столько же раз, сколько для этого потребуется владельцу устройства (в силу ошибки первого рода иногда ввод нужно будет повторять более 1 раза).

Кроме того, проверялась успешность аутентификации не только по целым паттернам, которые вводились, но и по их частям.

Эксперимент	Классификатор	FRR, %	FAR, %	Precision, %	Evil_Probability, %
1. Целая последовательность	SVM (1)	60.0	3.4	92.1	8.3
	SVM (2)	75.7	0.5	93.2	2.4
	Fuzzy	58.7	3.9	91.6	9.3
2. Первая половина фразы	KNN	63.7	3.6	91.5	9.6
3. Вторая половина фразы	Fuzzy (1)	58.7	1.2	94.1	9.4
	Fuzzy (2)	73.7	0.0	94.0	0.0

Таблица 2. *Последовательность 1*

Эксперимент	Классификатор	FRR, %	FAR, %	Precision, %	Evil_Probability, %
1. Целая последовательность	SVM	60.7	1.1	94.1	2.7
	KNN	64.5	7.5	87.9	9.3
	Fuzzy	68.3	1.2	93.4	3.7
2. Первая треть фразы	**SVM**	**53.1**	**0.9**	**94.8**	**2.1**
3. Вторая треть фразы	SVM	29.1	5.9	92.1	8.3
	Fuzzy	69.6	1.1	93.4	3.6
4. Третья треть фразы	SVM	50.6	3.6	92.5	7.2
	Fuzzy	69.6	1.9	92.5	6.4

Таблица 3. *Последовательность 2*

В таблицах выше (2 и 3) приведены не все результаты, а только лучшие по паре «образец – классификатор». Причём для некоторых образцов мог получиться выигрышным один классификатор, но с разными параметрами. И, наоборот, некоторые алгоритмы классификации для данного образца смотрелись гораздо проигрышнее, нежели его «конкуренты», поэтому в таблице они не приведены.

Заключение

На базе исследованных методов и разработанных скриптов классификации были отобраны две модели аутентификации, которые показали наилучшую точность в проведенных экспериментах. Первая на собранных данных вообще не допускает ошибки второго рода, но при этом владельцу смартфона, в среднем, придётся 4 раза ввести одну и ту же строку в 10 символов. Используя вторую модель, для доступа в систему необходимо два раза ввести уже другую строку в 10 символов,

при этом злоумышленник этот доступ получит только в одном случае из 50 (при том условии, что он вообще знает пароль). Конечно, на текущий момент обе эти модели вряд ли можно использовать в промышленных масштабах для защиты от взлома девайса с секретными данными, однако они могут выступать в качестве хорошей дополнительной защиты собственного смартфона от несанкционированного доступа.

Среди недостатков реализованного модуля можно выделить необходимость обучения приложения и сильную зависимость от эргономики и психофизического состояния пользователя смартфона.

В качестве дальнейшего развития проекта планируется протестировать реализованный прототип аутентификации на определённой группе человек, имеющих смартфоны на операционной системой iOS.

Литература

1. Syed Zulkarnain Syed Idrus, Estelle Cherrier «Soft Biometrics for Keystroke Dynamics», 2014.

2. Juan Liu, Baochang Zhang, Linlin Shen, Jianzhuang Liu, Jason Zhao. «The BeiHang Keystroke Dynamics Authentication System», 2011.

3. Romain Giot, Mohamad El-Abed, Christophe Rosenberger «Keystroke Dynamics With Low Constraints SVM Based Passphrase Enrollment», 2009.

4. A. Buchoux and N. Clarke “Deployment of keystroke analysis on a smartphone”, 2008.

5. Procedia Technology // 8th International Conference Interdisciplinarity in Engineering. «Keystroke Dynamics on Android platform», 2014.

6. Salima Douhou, Jan R. Magnus. «The reliability of user authentication through keystroke dynamics», 2009.

7. Ажмухамедов И. М., Варварина С. В. «Извлечение биометрических данных при рукописном вводе текста», 2008.

8. Steinwart, I., Christmann, A. “Support vector machines. Springer”, 2008.

9. Tubin G. “Emergence of Risk-Based Authentication in Online Financial Services”, 2005.

10. P. S. Teh, A. B. J. Teoh, T. S. Ong, and C. Tee “Performance enhancement on keystroke dynamics by using fusion rules”, 2008.

11. Giot, R., El-Abed, M. “Keystroke dynamics overview”, 2011.

12. Weijia Wang, Huanren Zhang, Aditi Gupta “Fuzzy Support Vector Machines”, 2014.

13. Matthias Trojahn, Frank Ortmeier “Biometric authentication through a virtual keyboard for smartphones”, 2012.

Попов И.С., Саликов П.М.

Исследование и разработка методов обнаружения аномалий в работе пользователя по данным системных и прикладных журналов

Введение

В данной работе рассматривается одна из актуальных задач компьютерной безопасности - задача обнаружения аномалий в работе пользователя.

В общем случае аномалия – это отклонение от нормы, от общей закономерности [1]. В случае работы пользователя с информационной системой можно выделить некоторые шаблоны поведения пользователя, которые называются ожидаемым поведением или моделью поведения пользователя. Эти шаблоны могут быть составлены как на основании предшествующего поведения пользователя, так и на основании поведения группы пользователей, в которой он состоит. Тогда аномалия в работе пользователя понимается как отклонение наблюдаемого поведения от ожидаемого поведения [2].

Обнаружение аномалий в работе пользователя позволяет решать актуальные задачи компьютерной безопасности, описанные далее.

Одной из таких задач является обнаружение подготовки к совершению внутренней атаки на информационную систему. При подготовке к совершению внутренней атаки человек, который ее совершает – инсайдер, вынужден провести подготовительные действия для сбора передаваемой информации и исследования возможностей ее передачи. Такие действия будут отличаться от его привычной работы, и поэтому являются аномалиями.

Отдельно стоит отметить такой тип внутренней атаки как подмена авторизованного пользователя. Так как поведение каждого человека индивидуально, то действия нового пользователя будут отличаться от действий авторизованного, что также является аномалией.

По результатам исследования [3] 67% специалистов по компьютерной безопасности считают, что их компании в какой-то степени подвержены внутренним угрозам безопасности. Также 62% специалистов отметили, что внутренние угрозы безопасности сложнее обнаружить, чем внешние, а средний ущерб составляет 445 000 долларов.

Ещё одной задачей, решению которой способствует обнаружение аномалий в работе пользователя, является обнаружение

использования рабочего времени в личных целях. Действия, совершаемые в личных целях, не характерны для нормального процесса работы, так что являются аномалиями в рабочем процессе.

По результатам опроса [4] самые типичные действия, совершаемые в личных целях в рабочее время – это чтение новостей, использование мессенджеров и социальных сетей, что отнимает в среднем от 24 до 46 минут рабочего времени. А посещение туристических сайтов и игры на компьютере хоть и более редки, но отнимают в среднем час рабочего времени.

Таким образом, решение задачи обнаружения аномалий в работе пользователя является актуальным, и позволяет достичь:

- предотвращения финансового ущерба от внутренних атак на информационную систему;
- предотвращения нарушения конфиденциальности, целостности и доступности данных в результате внутренней атаки на информационную систему;
- повышения эффективности труда благодаря сокращению использования рабочего времени в личных целях.

1. Описание решения

Основной целью данной работы является исследование и разработка методов решения задачи обнаружения аномалий в работе пользователя по данным системных и прикладных журналов, а также разработка на основе данных методов экспериментального прототипа системы обнаружения вторжений.

1.1. Обзор существующих методов решения задачи

Для выявления аномалий в работе пользователя сначала собираются некоторые данные о работе пользователя с информационной системой. По работе пользователя с устройствами ввода-вывода обычно собираются характеристики нажатия клавиш на клавиатуре: время между нажатиями клавиш, время удержания клавиш [5]. По работе пользователя с информационными и вычислительными ресурсами данные получают из событий операционной системы, журналов приложений и деталей сетевого трафика. Эти данные, обычно, собираются с помощью SIEM-систем (Security Information and Event Management — технология управления информацией и событиями безопасности) [6]. По работе пользователя с текстовой информацией в качестве параметров оценки часто выделяют основные тематики текста [7].

Далее из собранной информации выделяются некоторые параметры действий пользователя – *признаки*. Анализируя значения признаков с помощью тех или иных методов можно получить некоторые оценки признаков, то есть построить модель поведения пользователя. Применение модели к наблюдаемым данным позволяет произвести классификацию полученной информации на легитимную и нелегитимную.

Было проведено исследование популярных методов классификации данных для решения задачи обнаружения аномалий в работе пользователя с информационной системой. Рассматривались такие традиционные методы как решающее дерево [8], метод опорных векторов [9], генетические алгоритмы [10] и их комбинация [9]. По результатам исследования сделан ряд выводов:

- Традиционные методы классификации данных в решении задачи обнаружения вторжений обычно не позволяют добиться точности более 90%. Предполагается, что точность решения данной задачи может быть улучшена.
- Рассматриваемые методы не учитывают возможного изменения ожидаемого поведения пользователя и с их помощью невозможно анализировать динамику его изменения.

По результатам исследования [7] был выбран подход, основанный на анализе текстовой информации с помощью метода ортонормированной неотрицательной матричной факторизации, где в качестве математической модели используются временные ряды. Отметим, что прогнозирование временных рядов позволяет анализировать динамику изменения поведения пользователя. Поэтому данный подход был выбран для решения поставленной задачи.

1.2. Предлагаемый подход

Рассмотрим подробнее построение решения задачи обнаружения аномалий в работе пользователя с помощью подхода, основанного на анализе текстовой информации на базе метода ортонормированной неотрицательной матричной факторизации и прогнозирования временных рядов.

В настоящее время системные и прикладные журналы разнородны и содержат различную информацию о работе пользователя. Представление журналов в виде текстовой информации позволяет объединить разнородную информацию и информацию из различных источников и провести ее анализ единственным методом.

Для выделения *признаков* из входных данных воспользуемся тематическим моделированием системных и прикладных журналов в текстовом представлении. На основе выделенной из текстовых данных

информации можно построить модель поведения пользователя в виде временных рядов. Чтобы использовать временные ряды данные разбиваются на временные окна. Построенная модель применяется к наблюдаемым данным. Результатом применения будет служить мера аномальности, на основании значения которой можно провести классификацию полученных данных на легитимные и нелегитимные данные.

Решение рассматриваемой задачи с помощью выбранного метода может быть разбито на несколько подзадач:

- представление журналов в текстовом формате;
- построение и прогнозирование модели поведения пользователя;
- сопоставление наблюдаемых данных прогнозу и вычисление меры аномальности.

Рассмотрим подробнее решение каждой из подзадач.

1.2.1 Представление журналов в текстовом формате

На данном этапе полученные входные данные, записанные в журналы событий, необходимо перевести в текстовое представление для дальнейшего применения метода ортонормированной неотрицательной матричной факторизации.

Будем считать, что журналы представляют собой таблицы, где каждая строка определяет какое-нибудь событие, а каждый столбец является атрибутом события и все события в журнале упорядочены в хронологическом порядке. Чтобы сопоставить данным журналов несколько текстовых документов, разобьем все события, имеющиеся в журнале на последовательные группы - временные окна. Одно окно будет содержать или фиксированное количество событий, или события за равный отрезок времени.

Теперь для каждого временного окна поставим в соответствие один текстовый документ. Для этого каждое событие переведем в текстовую строку, записав последовательно через разделитель его атрибуты. Будем иметь в виду, что атрибуты событий могут быть числового или строкового типа. Строковые атрибуты запишем без изменения, а для числовых атрибутов с целью уменьшить количество различных значений проведем процесс дискретизации. Он состоит в следующем: все возможные значения числового параметра разбиваются на равновероятные интервалы, и вместо значения записывается номер одного или нескольких интервалов, в которое попало значение атрибута.

Таким образом, каждому событию сопоставляется строка – разделенные пробелом атрибуты, возможно преобразованные с

помощью дискретизации. Каждой последовательности событий – временному окну, сопоставляется последовательность строк, образующая текстовый документ. Всем событиям, предварительно разбитым на временные окна, сопоставлена последовательность документов, упорядоченная по времени, так как временные окна также упорядочены. Получено требуемое текстовое представление входных данных.

1.2.2 Построение и прогнозирование модели поведения пользователя

Из полученных текстовых документов с помощью тематического моделирования можно выделить основные признаки действий пользователя и соответствующие им веса в каждом временном интервале. Для каждого признака получим временной ряд, а временные ряды для всех признаков образуют матрицу A.

Для построения модели поведения пользователя разложим получившуюся матрицу в произведение двух, используя метод ортонормированной неотрицательной матричной факторизации $\mathrm{A} = W_K H_K$. Причем матрица H_K будет содержать временные ряды для объединённых в тематики признаков и рассматривается как модель поведения пользователя.

Для прогнозирования поведения пользователя для каждого временного ряда тематики спрогнозируем несколько следующих значений ряда с помощью решения задачи подстановки пропушенных значений.

Более подробно подход построения модели пользователя и прогнозирования рядов тематик с помощью выбранного метода описан в статье [7].

1.2.3 Сопоставление наблюдаемых данных прогнозу и вычисление меры аномальности

На данном этапе, получив данные от пользователей за прогнозируемый промежуток времени, мы должны сопоставить наблюдаемые данные и прогнозируемое поведение пользователя, чтобы классифицировать полученные данные как легитимные или нелегитимные.

Для этого переведем полученные события в текстовые документы, как это было сделано на этапе представления журналов в текстовом формате. Также как и при составлении модели представим действие пользователя в виде числовой матрицы A. Умножаем

матрицу A на W_K^T слева (матрица W_K получена на этапе построения модели пользователя) $W_K^T A = H_K^{`}$.

Получаем матрицу, содержащую веса ранее выделенных тематик для наблюдаемых данных. Заметим, что тут применено свойство ортогональности матрицы W_K^T ($W_K^T = W_K^{-1}$).

Теперь имея для выделенных тематик спрогнозированные и реальные веса необходимо рассчитать меру аномальности действий пользователя. Сопоставим каждому временному интервалу $0 \le i \le n$ значение аномальности от 0 до 1 равное нормированной сумме модулей отклонения спрогнозированного значения от реального значения по каждой тематике.

$$Anomaly_i = \Sigma_{j=1}^{m} | h_{ji}^{`} - h_{ji} | / m$$

Чем ближе полученное значение меры аномальности для конкретного временного интервала к единице, тем больше вероятность того, что в этот временной интервал с информационной системой работал не тот пользователь, по которому была составлена модель. И наоборот, чем ближе аномальность временного интервала к нулю, тем больше вероятность, что данные полученные за этот интервал принадлежат легитимному пользователю.

2. Программная реализация

Для решения поставленной задачи был разработан программный стенд, реализующий экспериментальный прототип системы обнаружения вторжений. В состав программного стенда входят три компоненты: локальный агент сбора данных, сервер консолидации данных и консоль анализа данных.

2.1 Архитектура экспериментального прототипа системы обнаружения вторжений

Общая архитектура экспериментального прототипа системы обнаружения вторжений изображена на рисунке 1.

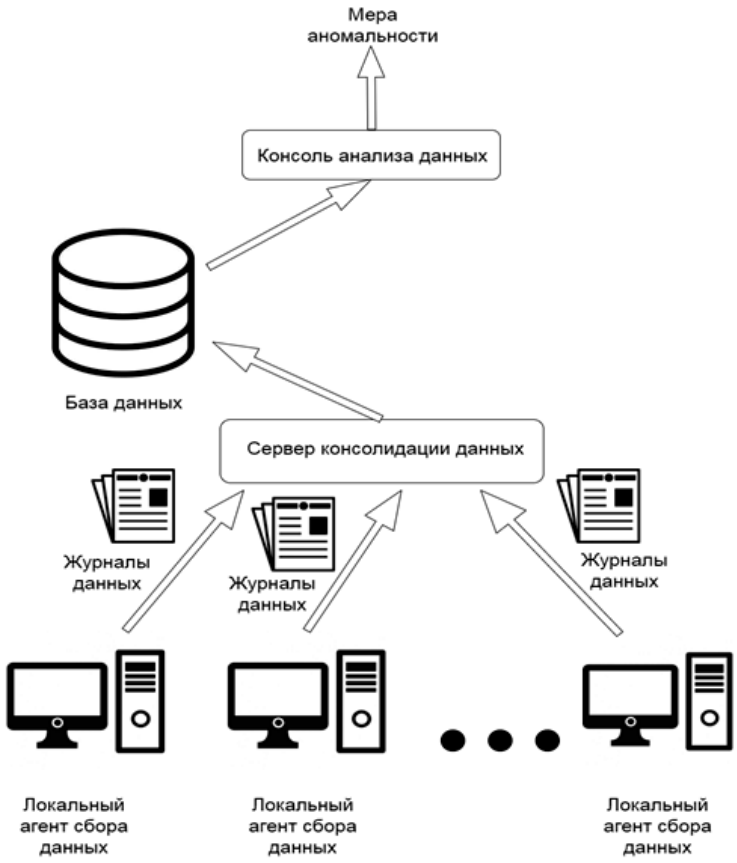

Рисунок 1. *Архитектура системы обнаружения вторжений*

Далее рассмотрим подробнее компоненты, входящие в состав экспериментального прототипа системы обнаружения вторжений.

2.2 Агент сбора данных

Для решения описанной выше задачи сбора данных о работе пользователя с системой и формирования журналов событий был взят за основу агент сбора данных, ранее разработанный в лаборатории. Агент был переработан с целью использования на новых операционных системах семейства Windows, включая поддержку 64битных операционных систем. Он реализован на языке C/C++ и состоит из нескольких основных компонент:

- Драйвер ядра файловой системы. Он собирает данные о работе пользователя с файловой системой, такие как обращение пользователя к файлам и папкам, операции с внешними запоминающими устройствами, использование CD/DVD. При разработке драйвера использовалась технология Windows Driver Development Kit.
- Службы Windows, осуществляющие подсчет действий с клавиатурой и мышью в единицу времени, а также собирающие данные о работе пользователя в сети
- Плагин для браузера Internet Explorer, который перехватывает и сохраняет отправленные формы.

Для накопления и хранения данных, собранных агентом локально был использован ранее разработанный в лаборатории сервер консолидации данных. Накопленные данные сервер преобразует в представление, пригодное для дальнейшего анализа и сохраняет их в базах данных.

2.3 Консоль анализа данных

Для решения задач построения модели пользователя и расчета аномальности была реализована консоль анализа данных. Консоль реализована на языке C# для операционных систем Windows версии 7 и выше. Пользовательский интерфейс реализован с помощью возможностей консоли Microsoft Management Console 3.0 и технологии Windows Forms. Также при разработке консоли использовались технологии Microsoft SQL и .NET 4.0.

Консоль анализа данных выполняет основные функции, описанные ниже.

- Построение выборки событий из журналов. На вход подаются настройки выборки (типы событий, атрибуты событий, временной интервал, дополнительные условия). На основе настроек формируются sql запросы к хранилищу, выбираются события и объединяются в общий список, упорядоченные по времени. На выходе имеем таблицу, содержащую запрашиваемые события.
- Построение моделей поведения пользователя по выборке. На вход подается обучающая выборка в виде таблицы событий и настройки модели. На основе настроек модели список событий обучающей выборки нарезается на временные окна, формируется матрица признаков. Матрица подается на вход алгоритму неотрицательной ортонормированной матричной факторизации, формируется модель поведения пользователя.

На выходе получаем модель поведения пользователя в виде временных рядов для выделенных алгоритмом тематик.

- Динамическое обновление моделей поведения пользователей. Для моделей, в настройках которых выбран пункт «Динамическое обновление модели» по расписанию производится перестроение модели пользователя. На выходе имеем модель с такими же параметрами, но с учетом новых событий.
- Прогнозирование поведения пользователя и сопоставление прогноза с реальными данными. Входными данными является модель поведения пользователя и тестовая выборка в виде таблицы событий, для которой на основе данной модели необходимо оценить меру аномальности. Модель поведения пользователя подается на вход алгоритма прогнозирования временных рядов, который считает прогноз по каждой тематике на заданное число шагов, а также формирует временной ряд отклонения от прогноза на основании тестовой выборки. На выходе имеем временные ряды тематик, выделенных в модели, для тестовой выборки и оценку меры аномальности для тестовой выборки.
- Демонстрация графиков, отображающих временные ряды для тематик и меру аномальности. На вход подается модель поведения пользователя или результат прогнозирования поведения пользователя. В результате для модели поведения пользователя визуализируются графики тематик, а для результата прогнозирования поведения пользователя также и графики прогнозов и аномальности.

3. Экспериментальное исследование

Для проверки качества предложенного метода решения рассматриваемой задачи на основе критериев оценки качества и оценок существующих методов был проведен ряд экспериментальных оценок. Экспериментальные оценки состояли из двух серий экспериментов, которые проводились на двух независимых наборах данных.

В первой серии экспериментов использовался открытый набор данных Киотского университета [11] (Kyoto 2006+ dataset). Набор данных содержит сетевые данные более чем 93 миллионов интернет сессий за период с 2006 по 2009 год. Сессии помечены как легитимные сессии - более 50 миллионов, сессий с известными типами атак - более 42 миллионов или сессии с неизвестными типами - атак более 400 тысяч. Для проведения экспериментов в данной работе были

использованы данные за февраль 2009 года. В экспериментах использованы признаки данных, описанные в статье [11].

Во второй серии экспериментов использовался набор данных, собранный в Лаборатории Технологий Программирования с помощью разработанного агента сбора данных. В наборе содержится повседневная работа 5 пользователей за период в 2 месяца. Все события помечены именем пользователя, который их совершил. В экспериментах использовались важные типы событий, описанные ниже:

- Запуск процессов. Данный тип событий характеризует взаимодействие пользователя с операционной системой и включает в себя информацию об имени и длительности процесса, а также об имени родительского процесса, который его запустил.
- Авторизация. Данный тип событий характеризует взаимодействие пользователя с системой авторизации и содержит информацию о фактах ввода пользователем учетных данных, в том числе процесс, запросивший учетные данные, информацию об успешности ввода, длительность работы под учетными данными.
- Работа в сети. Данный тип событий характеризует взаимодействие пользователя с локальными и глобальными сетями и содержит факты передачи и получения данных по сети. Факты включают в себя информацию об объеме переданных или полученных данных, адресе и порте отправителя и получателя, процессе, в рамках которого данные были переданы и протоколе передачи данных.
- Работа с клавиатурой и мышью. Данный тип событий характеризует взаимодействие пользователя с такими устройствами ввода данных как клавиатура и манипулятор мышь и включает в себя информацию о количестве нажатий клавиатуры и действий мышью за некоторый промежуток времени.
- Данные файловой системы. Данный тип событий характеризует взаимодействие пользователя с файловой системой и содержит события чтения файлов и записи в файл. События включают в себя информацию о процессе, совершившем операцию, объеме данных, имени файла и типе носителя, с которого данные были прочитаны или на который данные были записаны.

3.1 Критерии оценки результатов

В качестве критериев оценки качества результатов экспериментов в данной работе использовались несколько показателей.

Точность решения, а также *процент ошибок первого рода* (*FAR – false acceptance rate*) и *процент ошибок второго рода* (*FRR – false rejection rate*) были выбраны как самые распространенныk и использующийся в статьях, описывающих традиционные методы решения рассматриваемой задачи [8, 9, 10].

Еще один показатель качества результатов, использующийся в данной работе – *площадь под ROC-кривой* (*ROC AUC - Rreceiver Operating Characteristic Area Under Curve*). Данный показатель был выбран для оценки качества решения, так как в отличие от описанных выше показателей учитывает разнородность тестовых данных.

3.2 Результаты экспериментов

Итак, по описанным выше данным был проведен ряд экспериментальных оценок. В первой серии экспериментов данные были разделены на обучающую и тестовую выборки в соотношении 2:1. Для построения модели пользователя использовались легитимные данные из обучающей выборки, которым затем сопоставлялись легитимные и нелегитимные данные из тестовой выборки. Заметим, что в данной серии экспериментов модель строится по группе пользователей, с целью определить отклонение от модели в нелегитимных данных. Исследовалась зависимость от размера временного окна событий, и оценивались *площадь под ROC-кривой*, *точность*, и *процент ошибок первого и второго рода*. Результаты для наилучших размеров окон представлены в Таблице 1.

Размер окна	Roc auc	Точность	Ошибка I рода	Ошибка II рода
20.000 событий	0.607	61.4%	5.6%	60.7%
25.000 событий	0.996	97.1%	0.0%	4.8%
30.000 событий	0.995	96.7%	0.0%	5.6%

Таблица 1. *Зависимость показателей качества от размера временного окна в первой серии экспериментов*

Во второй серии экспериментов данные были разделены на обучающую и тестовую выборки в соотношении 5:1. Для построения модели пользователя использовались данные из обучающей выборки соответствующего пользователя, которым затем сопоставлялись данные из тестовой выборки всех пользователей. Целью служит определить отличие от модели поведения пользователя в поведении других

пользователей. Исследовалась зависимость от размера временного окна событий и выбранных признаков, и оценивались *площадь под ROC-кривой*. Результаты для наилучших размеров окон и наборов признаков представлены в Таблице 2.

Учитываемые признаки	Размер окна	Roc auc
Все признаки	750 событий	0.852
Все признаки	1000 событий	0.946
Все признаки	1500 событий	0.750
Без имени файла и процесса	750 событий	0.909
Без имени файла и процесса	1000 событий	0.982
Без имени файла и процесса	1500 событий	0.700

Таблица 2. *Зависимость площади под ROC-кривой от размера временного окна и выбранных признаков во второй серии экспериментов*

Для моделей с лучшими параметрами была также исследована зависимость *площади под ROC-кривой* от размера тестовой выборки. Результаты представлены в Таблице 3.

Длина прогноза	1/5 обучающей	1/3 обучающей	1/2 обучающей	1 обучающая
Лучшая модель	0.982	0.672	0.700	0.711

Таблица 3. *Зависимость площади под ROC-кривой от размера тестовой выборки во второй серии экспериментов*

3.3 Выводы из экспериментов

По результатам проведенных экспериментальных оценок можно сделать ряд выводов:

- На основе критериев оценки качества и оценок существующих методов предложенный метод решения задачи показал высокие результаты.
- Выбранный метод применим как для построения модели поведения отдельного пользователя, так и группы пользователей.
- Размер временного окна для наилучшей модели зависит от размера обучающей выборки и должен выбираться пропорционально её размеру
- Качество выбранного метода зависит от количества событий в тестовой выборке и тем лучше, чем меньше событий в ней. Это связано с тем, что в выбранном методе используется

прогнозирование временных рядов, и качество прогноза уменьшается с увеличением длительности.

Заключение

Главными итогами данной работы являются следующие результаты:

- по результатам исследования существующих методов обнаружения аномалий в работе пользователя предложен собственный метод, основанный на отрицательной ортонормированной матричной факторизации и прогнозировании временных рядов;
- на основе выбранного метода разработан экспериментальный прототип системы обнаружения вторжений;
- с помощью разработанного программного обеспечения проведена экспериментальная оценка качества выбранного метода, и на основе критериев оценки качества и оценок существующих методов выбранный метод решения задачи показал ожидаемо высокие результаты.

Литература

1. (2017, апрель) Веб-сайт "Академик". [Online]. http://dic.academic.ru/dic.nsf/enc_philosophy/1750/%D0%90%D0%9D%D0%9E%D0%9C%D0%90%D0%9B%D0%98%D0%AF
2. Chandola V., Banerjee A., Kumar V. Anomaly detection: A survey //ACM computing surveys (CSUR). – 2009. – Т. 41. – №. 3. – С. 15.
3. (2017, апрель) Веб-сайт компании "Palerra". [Online]. https://palerra.com/cloud-security-resources-2/infographics/
4. (2017, апрель) Веб-сайт "Pandia". [Online]. http://pandia.ru/text/78/238/57269.php
5. Killourhy K. S., Maxion R. A. Comparing anomaly-detection algorithms for keystroke dynamics //Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on. – IEEE, 2009. – С. 125-134.
6. Thakare S., Ingle P., Meshram B. B. IDS: Intrusion Detection System the Survey of Information Security //VJTI, Matunga, Mumbai. – 2012. – Т. 2.
7. В.Ю. Королёв , А.Ю. Корчагин, И.В. Машечкин, М.И. Петровский, Д.В. Царёв Применение временных рядов в задаче фоновой идентификации пользователей на основе анализа их работы с текстовыми данными // Труды Института системного программирования РАН - Том 2.

8. Wu S. Y., Yen E. Data mining-based intrusion detectors //Expert Systems with Applications. – 2009. – Т. 36. – №. 3. – С. 5605-5612.
9. Peddabachigari S. et al. Modeling intrusion detection system using hybrid intelligent systems //Journal of network and computer applications. – 2007. – Т. 30. – №. 1. – С. 114-132.
10. Hoque M. S. et al. An implementation of intrusion detection system using genetic algorithm //International Journal of Network Security & Its Applications, Volume 4, Number 2, pages 109 - 120, March 2012.
11. Song J. et al. Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation //Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security. – ACM, 2011. – С. 29-36.

Королев Л.Н.

Архитектура и программирование

Автору в течение почти 50 лет (с 1953 года) посчастливилось быть свидетелем и непосредственным участником разработки структур и программного обеспечения некоторых отечественных вычислительных систем, разработанных под руководством Сергея Алексеевича Лебедева. Суждения автора, хотя и носят субъективный характер, могут быть интересны современным компьютерщикам.

Проектировщики вычислительных систем сейчас и раньше стремились достичь одновременно двух целей:

- из имеющейся в наличии элементной базы построить вычислитель с наилучшим соотношением «стоимость / производительность» для данного класса потенциальных пользователей;
- сделать машину удобной для программирования.

Первые экземпляры вычислительной техники у нас и за рубежом ориентировались на узкий класс пользователей, в основном физиков и ракетчиков, создававших атомное оружие и средства его доставки. Главным было достичь, не обращая внимание на затраты, по возможности высокой производительности и приемлемой надежности.

К этому типу по существу уникальных машин у нас следует отнести «Стрелу» и «БЭСМ-1».

Роль математиков, обслуживавших физиков и ракетчиков, на этапе проектирования этих машин поначалу состояла в определении минимально необходимой разрядности машинного слова для хранения операндов, с учетом требуемой точности результатов вычислений, и в предложениях по кодировке полей машинных команд, что можно назвать разработкой машинного языка.

Как сейчас представляется, 50-е годы явились началом новой ветви математики, получившей название прикладной математики, тесно связанной с построением алгоритмов, доступных для реализации на вычислительной машине.

Потребовались новые подходы к теоретическим исследованиям в области численных методов в условиях ограничений разрядной сетки реальных машин, потребовалось исследование феномена программ, совершенно новых объектов в математике, правил взаимодействия и формального преобразования программ, изучение эквивалентности и сложности программ. Потребовалась разработка новых эффективных алгоритмов и доказательства их работоспособности, правильности получаемых результатов. Именно на основе этих исследований

математиками предлагались и обосновывались требуемые характеристики проектируемых вычислительных машин.

В начале 50-х годов под грифом «секретно» вышла замечательная книга, в которой описаны многие оригинальные вычислительные алгоритмы, пригодные для реализации на ЭВМ. Среди ее авторов были такие выдающиеся математики как М.Р. Шура-Бура, А.А. Абрамов, М.А. Люстерник и др. Эта книга оказала большое влияние на развитие программирования в СССР, сформировав взгляд на эту отрасль знаний, как на новую часть математики вообще, и прикладной математики в частности.

«Удобство» программирования, которое учитывается при проектировании ЭВМ, понятие достаточно расплывчатое, смысл которого значительно изменялся со временем, так же как и само понятие программирования.

Вначале на этапе программирования в машинных кодах под удобством программирования понималась хорошая мнемоника машинного языка в 8-ричном или 16-ричном цифровом представлении, в зависимости от перфорационного оборудования.

С появлением алфавитно-цифровых перфораторов и первых языков символьного кодирования (автокодов, ассемблеров) роль внутренней кодировки перестала играть существенную роль. Приобрели значение запоминающаяся мнемоника записи смысла операций, символика задания имен (символических адресов) переменных. Синтаксис первых машинно-независимых языков типа Фортран и Кобол явно несет отпечатки синтаксиса автокодов.

Первым машинно-независимым языком с синтаксисом и семантикой, приближенными к математической нотации, по-видимому, следует считать Алгол-60, изначально предназначавшийся для записи и публикации «математических» алгоритмов. В Советском Союзе он в свое время был возведен в ранг стандарта.

Первые трансляторы с Алгола-60 были созданы в СССР, что явилось крупным достижением того времени. Две конкурировавшие разработки были выполнены под руководством и при непосредственном участии выдающихся математиков М.Р. Шуры-Буры и С.С. Лаврова.

Свойства блочно- и процедурно-ориентированных языков типа Алгол нашли свое отражение в архитектурах некоторых машин, а именно, в машинах фирмы Барроуз (Burroughs) и в отечественных машинах серии Эльбрус. Это пример влияния языкотворчества на архитектуру ЭВМ, выразившегося в создании аппаратной поддержки таких удобных средств программирования, как рекурсии и процедуры.

ЭВМ начала 50-х годов, как уже было сказано, были созданы на потребу военно-промышленных комплексов и на поддержку научных исследований, так или иначе связанных с ними. Несмотря на это, на этих машинах проводились инициативные экспериментальные работы,

носившие чисто научный, академический характер. К заслугам С.А. Лебедева следует отнести то, что он давал возможность математикам работать в этих направлениях, вообще говоря, далеких от суровых правительственных заданий на создание необходимых государству машин и решение «государственных» задач.

В частности, на БЭСМ-1 проводились эксперименты по автоматическому переводу с английского языка на русский. В составе коллектива, работавшего под руководством Д.Ю. Панова и при кураторстве зам. директора ИТМ и ВТ И.С. Мухина, над проблемами машинного перевода довелось работать и автору доклада, предложившему и реализовавшему механизм для ускорения поиска по словарю, получивший потом название «хеширование». Чисто академический интерес представляли первые работы по символьному дифференцированию, выполненные при участии и руководстве автора на той же машине БЭСМ-1, совсем не предназначенной для решения задач подобного типа. В БЭСМ-1 отсутствовали операции символьной обработки, были только операции арифметики с плавающей запятой и минимальный набор поразрядных логических операций.

Обусловленная коммерческими и научными интересами погоня за универсальностью ЭВМ, в смысле обработки как можно более широкого спектра типов данных, естественно привела к резкому усложнению системы команд и логики работы аппаратуры. Так появились универсальные вычислительные машины со сложной системой команд, машины CISC-архитектуры, в которых аппаратно поддерживается большое разнообразие типов данных и операций над ними. Логическую сложность машины характеризует число различных команд. В системе IBM 360/370, например, это число приближалось к 300. Создание машин CISC-архитектуры стало модным.

Это привело к необычайному усложнению процессора и вошло в противоречие с «инженерной» целью – добиться максимальной производительности аппаратуры при решении конкретных задач.

Думается, что уверенность С.А. Лебедева в возможности и потенциале отечественных математиков позволили ему не поддаться этой моде, задержавшей внедрение новых идей в организацию машин на некоторое значительное время.

Следует отметить, что консервативная идея сохранить стиль работы пользователя не выдержала испытание временем. Пользователю непрерывно приходится доучиваться или переучиваться, если он хочет воспользоваться новыми мультимедийными возможностями, сетевым взаимодействием, компьютерной телефонией и другими новейшими услугами компьютерных систем.

Многими программистами была высоко оценена прозрачная и логически простая система команд отечественной машины БЭСМ-6, архитектура которой отражала гениальный стиль мышления ее главного

конструктора, выдающегося ученого, патриарха отечественной вычислительной техники, академика Сергея Алексеевича Лебедева.

Огромный успех этой машины среди пользователей обязан простоте ее логического построения, простоте ее архитектуры. Автор гордится тем, что принимал непосредственное участие в разработке машины БЭСМ-6, и считает себя одним из многих учеников Сергея Алексеевича Лебедева.

Процессоры RISC-архитектуры, к которым можно отнести и БЭСМ-6, такие как Alpha-DEC, POWER PC, SUN-SPARC, HP-PA, в большей степени ориентированные на сферу применения в научных исследованиях, решения академических задач, приобрели в настоящее время доминирующее значение.

У Сергея Алексеевича Лебедева был прекрасный принцип внимательного отношения к инициативам молодых исследователей, предлагавших часто нереализуемые в данное время проекты, касающиеся новаций в архитектурах ЭВМ. Он внимательно выслушивал проектантов и в их присутствии рассматривал все «за» и «против», не умаляя достоинств, и не подчеркивая заблуждения.

Сергей Алексеевич Лебедев высоко оценивал значение математиков и программистов, от которых во многом зависели успехи применения разработанных в Институте вычислительных машин в сложнейших системах управления в реальном масштабе времени, в частности в системах противоракетной обороны. Он сам пробовал программировать некоторые задачи, думается, главным образом для того, чтобы оценить трудоемкость и значимость этого процесса на основании собственного опыта.

Может быть поэтому он оставлял без административных последствий некоторые ошибки программистов, которые доставляли ему как Главному конструктору очевидные неприятности.

Аннотации

Смелянский Р.Л. ИКТ инфраструктура нового поколения и задачи научно-образовательной сферы // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

Статья посвящена анализу трендов в области информационно телекоммуникационных технологий. Обосновывается актуальность основные цели, тематика и ожидаемые результаты проекта проекта GRANIT.

Ил.: 1 рис.

Аграновский М.Л., Шалимов А.В. Применение хеш-функций для ускорения работы программных OpenFlow-коммутаторов // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В статье рассмотрена система кеширования наиболее часто исполняемых правил программного OpenFlow-коммутатора с множественными хэш-таблицами, взаимно-однозначно связанными с таблицами потоков коммутатора. Для нее предложен метод повышения производительности, основанный на учете полей сравнений, используемых OpenFlow-приложениями, функционирующими в каждой конкретной таблице потоков.

Ил.: 5 рис., 1 табл. Библиогр.: 8.

Пашков В.Н., Гуськов Д.А. Метод выбора отказоустойчивого размещения контроллеров в глобальных программно-конфигурируемых сетях // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В работе рассматривается проблема размещения контроллеров распределённой платформы управления в глобальных программно-конфигурируемых сетях. Основными критериями оптимального размещения является минимизация стоимости обеспечения управления и минимизация средней задержки на установление нового потока в сети. Дополнительным требованием является обеспечение устойчивости сети к отказу не более одного контроллера и не более одного коммутатора. Для решения поставленной задачи встатье предлагается жадный алгоритм, включающий в себя алгоритмы на графе и решение задачи булевского линейного программирования, разрабатывается программное средство и приводятся результаты экспериментального исследования разработанного алгоритма.

Ил.: 4 рис., Библиогр.: 11.

Смелянский Р.Л., Слюсарь Д.Р. Обзор протоколов надежной многоадресной рассылки для применения в задаче управления беспроводными локальными Wi-Fi сетями // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В данной статье исследованы транспортные протоколы, поддерживающие надежную многоадресную рассылку, с целью поиска оптимального протокола для организации централизованной рассылки контрольных сообщений с минимальными затратами. По результатам обзора литературы были выделены наиболее подходящие протоколы, впоследствии протестированные на экспериментальном стенде средства для конфигурирования Wi-Fi сетей.

Ил.: 3 рис., 1 табл. Библиогр.: 10.

Швецов Д.А., Шалимов А.В. Разработка системы автоматической генерации правил в программно-конфигурируемых сетях // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В статье исследуются накладные расходы систем автоматической генерации правил и предлагаются методы по их устранению.

Ил.: 6 рис., 1 табл. Библиогр.: 6.

Шемякин Р.О., Петров И.С. Обеспечение контроля доступа приложений к ресурсам контроллера программно конфигурируемых сетей // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В статье приведен сравнительный анализ существующих решений по обеспечению контроля доступа приложений к интерфейсам управления сетевыми устройствами на SDN контроллере, приведено описание разработанной системы контроля доступа к интерфейсам управления сетевыми устройствами и ее экспериментальное исследование.

Ил.: 5 рис., 1 табл. Библиогр.: 16.

Шендяпин А.С., Петров И.С. Исследование методов проведения атаки Man-in-the-Middle в программно-конфигурируемых сетях // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В данной статье проводится исследование уязвимости программно-конфигурируемых сетей к атакам Man-in-the-middle, реализуемых с использованием скомпрометированного коммутатора. Приводятся разработанные методы проведения данных атак в различных сетевых сценариях. Разработанные методы реализуются в виртуальной сети с использованием Mininet под управлением SDN контроллера Ryu.

Ил.: 5 рис., 1 табл. Библиогр.: 11.

Костенко В.А, Чупахин А.А. Влияние миграции на эффективность использования ресурсов центров обработки данных // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В статье приведены результаты экспериментального исследования влияния миграции на эффективность планирования ресурсов в центре обработки данных по критериям количество размещенных запросов и загрузка серверов.

Ил.: 2 рис., 2 табл. Библиогр.: 4.

Костенко В.А., Шостик А.В. Алгоритмы имитации отжига для задачи отображения виртуальных ресурсов на физические в центрах обработки данных // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В статье приведена постановка задачи отображения виртуальных ресурсов на физические в центрах обработки данных, предложен алгоритм имитации отжига для решения данной задачи и приведены результаты экспериментального исследования его свойств.

Ил.: 3 рис., Библиогр.: 9.

Пинаева Н.М., Антоненко В.А. Разработка и реализация системы управления версиями виртуальных сетевых функций в облачной платформе // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В работе рассмотрена задача разработки системы управления версиями виртуальных сетевых функций в облачной платформе, приведено описание разработанной политики обновления, системы обновления виртуальных сетевых функций и экспериментального исследования работы данной системы.

Ил.: 3 рис., 3 табл. Библиогр.: 8.

Антипина Е.А., Балашов В.В. Обобщенная задача о мультипликативном рюкзаке со связями между объектами // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В статье представлен новый тип задачи о рюкзаке — обобщенная задача о мультипликативном рюкзаке со связями между объектами, возникающая, в частности, при распределении вычислительной нагрузки в модульных вычислительных системах. Также представлен алгоритм решения этой задачи, основанный на методе ветвей и границ. Отличительной чертой алгоритма является подход к вычислению верхних и нижних оценок на подмножествах области решений.

Ил.: 1 рис. Библиогр.: 7.

Малышев Н.С., Волканов Д.Ю. Средство анализа истории изменений текста программы // Программные системы и инструменты. Тематический сборник №17, М.: Изд-во факультета ВМиК МГУ, 2017.

В статье приведена постановка задачи анализа множества версий программы на предмет клонирования фрагментов исходного кода, содержащих ошибки и уязвимости, приведено описание алгоритма поиска клонов кода обнаруженных фрагментов, содержащих ошибки, при помощи представления фрагментов исходного кода в виде абстрактных синтаксических деревьев и его экспериментальное исследование.

Ил.: 3 рис., 2 табл. Библиогр.: 21.

Горохов О.Е., Петровский М.И., Машечкин И.В. Применение сверточных нейронных сетей для задач обнаружения аномалий в работе пользователя с текстовыми данными // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМиК МГУ, 2017.

В данной статье рассматривается проблема обнаружения аномалий в текстовых данных с использованием сверточных нейронных сетей. В данной статье предлагается архитектура сверточной сети, позволяющая решить эту проблему. В статье приведено описание предложенного и реализованного алгоритма с обратной связью для построения статико-динамического расписания, а также результаты экспериментальной апробации алгоритма.

Ил.: 2 рис., 1 табл. Библиогр.: 9.

Закляков Р.Д., Петровский М.И., Попов И.С. Разработка гибридных методов распознавания пользователя по особенностям его работы со стандартными устройствами ввода компьютера // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМК МГУ, 2017.

В рамках данной статьи разработаны специальные сценарии работы, которые можно интегрировать в повседневный цикл взаимодействия пользователя с компьютером, побуждающие его произвести заранее предопределенную последовательность действий. Предложены методы анализа характера прохождения пользователем указанных сценариев, включая методы построения обучающей выборки на основе данных фонового сбора, методы отображения событий, возникающих при использовании устройств ввода, в признаковые пространства, а так же методы одноклассовой классификации. Предложен метод оценки отношения выбросовых векторов в обучающей выборке к их общему числу. Проведена экспериментальная оценка качества предложенных сценариев и методов анализа характера их прохождения.

Ил.: 5 рис., 2 табл. Библиогр.: 9.

Красняков Е.И., Попов И.С. Аутентификация пользователя мобильного устройства по характеристикам работы с виртуальной клавиатурой // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМК МГУ, 2017.

В этой статье рассматривается аутентификация пользователей по их манере ввода символов с экранной клавиатуры. На базе исследованных методов и разработанных скриптов классификации были отобраны две модели аутентификации, которые показали наилучшую точность в проведенных экспериментах. В качестве дальнейшего развития проекта планируется протестировать реализованный прототип аутентификации на определённой группе человек, имеющих смартфоны на операционной системой iOS.

Ил.: 3 рис., 3 табл. Библиогр.: 13.

Попов И.С., Саликов П.М. Исследование и разработка методов обнаружения аномалий в работе пользователя по данным системных и прикладных журналов // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМК МГУ, 2017.

В данной статье рассматривается задача обнаружения аномалий в работе пользователя. По результатам исследования существующих методов обнаружения аномалий в работе пользователя предложен собственный метод, основанный на отрицательной ортонормированной матричной факторизации и прогнозировании временных рядов, на основе выбранного метода разработан экспериментальный прототип системы обнаружения вторжений. С помощью разработанного программного обеспечения проведена экспериментальная оценка качества выбранного метода.

Ил.: 1 рис., 3 таб. Библиогр.: 11.

Королев Л.Н. Архитектура и программирование // Программные системы и инструменты. Тематический сборник № 17, М.: Изд-во факультета ВМК МГУ, 2017.

В данной статье автором приводятся суждения, которые хотя и носят субъективный характер, могут быть интересны современным компьютерщикам. Эти суждения посвящены тому как проектировщики вычислительных систем пытаются достичь одновременно двух целей из имеющейся в наличии элементной базы построить вычислитель с наилучшим соотношением «стоимость / производительность» для данного класса потенциальных пользователей и сделать машину удобной для программирования

Software systems and tools: Periodical / Ed. R.L. Smelyansky. – M: Publishing Department of the Faculty of Computational Mathematics and Cybernetics (license ID № 05899 from 24.09.2001); MAKS Press, 2017. – № 17. – 200 p.

Theese proceedings consists of student's works, who have received the recommendation of the Department's scientific seminars, which he created and directed Korolev L. This edition continues the tradition of publishing in memory of this outstanding man. The proceedings contains articles devoted to creating information computing infrastructure for scientific research, problems of modern computer networks, methods and tools for organizing and managing cloud computing, tools that support the operation of real-time systems (SRV), methods for user authentication and the analysis of their behavior in computer networks in the interests of information security.

These papers will be of interest to students, graduate students and professionals in the development of application software systems using new information technologies.

Keywords: information telecommunication technology, software-defined networking, OpenFlow switch, centralized controller, wireless Wi-Fi networks, application access control, computer security attack, cloud computing, data center, simulated annealing, virtual resources, network functions virtualization, cloud platform, knapsack problem, real-time systems, integrated modular avionics, job scheduling, source code clones, software tools, information security, convolutional neural networks, user authentication problem, hybrid user recognition methods, computer architecture, programming.

Научное издание

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 17

Под общей редакцией
чл.-корр. РАН, профессора Р.Л. Смелянского

Напечатано с готового оригинал-макета
Подписано в печать 06.12.2017 г.
Формат 60x90 1/16. Усл.печ.л. 12,5. Тираж 100 экз. Заказ 293.
Издательство ООО “МАКС Пресс”
Лицензия ИД N 00510 от 01.12.99 г.
119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус, 527 к.
Тел. 8(495)939-3890/91. Тел./Факс 8(495)939-3891.