МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 16

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

МАКС ПРЕСС

МОСКВА – 2016

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, 2016. – № 16. – 148 с.
ISBN 978-5-89407-565-5
ISBN 978-5-317-05358-1

Данный сборник посвящен памяти Л.Н. Королёва. Он составлен по материалам работ студентов, получивших рекомендации научных семинаров кафедры, которую он создал и бессменно руководил. В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные описаниям инструментальных программных систем, разработанных авторами публикаций, относящихся к методам программирования для супер-ЭВМ, анализу текстов на естественных языках, математической лингвистике, инструментальным средствам, обеспечивающим работу систем управления в реальном времени (СРВ) и некоторым вопросам теоретического программирования. В нем также публикуются статьи и заметки общего характера, касающиеся разделов информатики, тесно связанных с программированием.

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

Ключевые слова: программно-конфигурируемые сети, сетевая операционная система, компьютерная атака, централизованный контроллер, информационно-ориентированные сети, облачные вычисления, виртуализация сетевых функций, имитационное моделирование, системы реального времени, интегрированная модульная авионика, синхронизация времени, построение расписаний, задача оптимизации надёжности, генетические алгоритмы, простые числа, факторизация больших чисел.

УДК 519.6+517.958
ББК 22.19

ISBN 978-5-89407-565-5
ISBN 978-5-317-05358-1

СОДЕРЖАНИЕ

СБОРНИК

«Программные системы и инструменты»

От редколлегии

Ушел из жизни Лев Николаевич Королев – талантливый учёный, увлёкший многих из нас своим азартом научных исследований, воспитавший десятки кандидатов и докторов наук. Его пионерские работы положили начало многим научным направлениям в области системного программирования: первые работы по применению ЭВМ для перевода с английского языка, применение ЭВМ для управления противоракетными системами, системное программное обеспечение для отечественных ЭВМ и многое, многое другое. Он внес огромный вклад в становление Программирования как учебной дисциплины. Его друзьями, учениками и сослуживцами составлен сборник воспоминаний о нем — Лев Николаевич Королёв: Биография, воспоминания, документы / Сост. и ред. Власов В.К., Смелянский Р.Л., Томилин А.Н. — М.: МАКС Пресс, 2016. — 272 с. [8 с. вкл.] ISBN 978-5-317-05333-8.

Тематический сборник “Программные системы и инструменты” был инициирован Львом Николаевичем в 2001 году, как площадка где студенты и молодые ученые могли бы публиковать свои достижения. Все эти годы Лев Николаевич неустанно вел его, отбирал публикации, редактировал.

Этот выпуск сборника посвящен его памяти. Он составлен по материалам работ студентов, получивших рекомендации научных семинаров кафедры, которую он создал и бессменно руководил. В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные описаниям инструментальных программных систем, разработанных авторами публикаций, относящихся к методам программирования для супер-ЭВМ, анализу текстов на естественных языках, математической лингвистике, инструментальным средствам, обеспечивающим работу систем управления в реальном времени (СРВ) и некоторым вопросам теоретического программирования.

В нем также публикуются статьи и заметки общего характера, касающиеся разделов информатики, тесно связанных с программированием.

Редколлегия

Раздел I.
Программно-Конфигурируемые Сети

Скобцова Ю.А., Пашков В.Н.

Исследование и разработка приложения мониторинга состояния распределенной платформы управления в программно-конфигурируемых сетях[1]

Введение

На сегодняшний день программно-конфигурируемые сети (ПКС) [1], [2] – активно развивающаяся технология, появление которой позволило выйти за пределы ограничений традиционных сетей за счет возможности гибкого конфигурирования компьютерной сети. В отличие от традиционных сетей, где сетевые устройства решают одновременно и задачу управления сетью, и задачу передачи данных, ПКС подразумевает разделение этих процессов: функция передачи данных остается в ведении сетевых устройств, в то время как функция управления устройствами и потоками данных передается специальному программному обеспечению – сетевой операционной системе (или контроллеру). Такое разделение значительно упрощает управление сетью, которое в ПКС сводится к написанию сетевых приложений для контроллера.

Однако централизованное управление сетью делает ПКС-контроллер единой точкой отказа: выход из строя контроллера может привести к нарушению работы пользовательских сетевых сервисов и отказу всей сети. Одним из решений проблемы является создание распределенной платформы управления ПКС – набора серверов, на каждом из которых запущен отдельный контроллер, при этом контроллеры взаимодействуют между собой и поддерживают общее представление сети. Контроллеры образуют распределенную сетевую ОС (РСОС). В настоящее время ведется активная разработка РСОС – это проекты ONOS [3], HAC [4], OpenDaylight [5], RunOS [7,8].

Распределенная платформа управления решает проблему существования единой точки отказа: если произойдет отказ какого-либо

[1] Проект поддержан грантом Инновационного фонда «Сколково» от 2 июля 2012г. №79.

из контроллеров, другие контроллеры РСОС смогут восстановить его работу за счет своих ресурсов (например, установка соединения с коммутаторами, связь с которыми потеряна, и возобновление управления ими). Тем не менее, с переходом к РСОС нагрузка на каждый отдельный контроллер не уменьшается. Негативное влияние на работу контроллера могут оказывать: сетевые устройства (например, обработка большого числа приходящих с коммутаторов сообщений), сетевые приложения (например, запуск большого числа ресурсоемких приложений на одном контроллере может привести к замедлению и нестабильности его работы), другие контроллеры, сетевые администраторы, злоумышленники. Поэтому владение информацией о нагрузке на контроллер РСОС позволит эффективно распределять нагрузку между узлами платформы (например, размещать ресурсоемкие приложения на менее нагруженных контроллерах), а также обнаруживать угрозы возможных перегрузок, отказов и безопасности контроллера и прогнозировать их. Таким образом, актуальной является задача мониторинга характеристик контроллера распределенной платформы управления ПКС в реальном времени. Стоит отметить, что существующие РСОС (ONOS, HAC и OpenDaylight) не поддерживают инструментов для такого мониторинга.

Данная работа посвящена исследованию и разработке приложения для мониторинга состояния контроллера в режиме реального времени. В качестве контроллера для реализации приложения используется OpenFlow-контроллер RUNOS, на базе которого строится распределенная платформа управления.

1. Особенности мониторинга состояния контроллера распределенной платформы управления

Типичная структура ПКС-контроллера представлена на рис. 1 [4]. Ядро контроллера реализует обработку приходящих с сетевых устройств сообщений, а также управление сетевыми устройствами, поддерживает представление сети (например, топологию).

Сетевые приложения взаимодействуют с контроллером через Northbound API – специализированный программный интерфейс, предоставляемый контроллером. В свою очередь, контроллер взаимодействует с коммутаторами через Southbound API – интерфейс для взаимодействия с сетевыми устройствами. OpenFlow – наиболее широко распространенный протокол взаимодействия ПКС-контроллера и сетевых устройств. В работе рассматривается контроллер, поддерживающий протокол OpenFlow версии 1.3 [6].

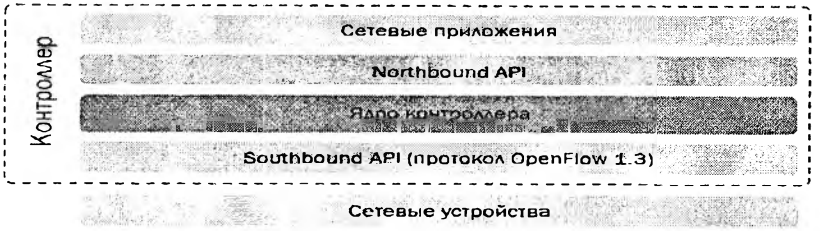

Рис. 1. Структура ПКС-контроллера с поддержкой OpenFlow

Рассмотрим подробнее механизмы взаимодействия OpenFlow-контроллера и сетевых устройств.

Протокол OpenFlow поддерживает три типа сообщений: асинхронные сообщения (от коммутатора контроллеру), сообщения контроллера коммутатору и симметричные сообщения. Асинхронные сообщения включают в себя Packet-in сообщения (неизвестный коммутатору пакет помещается в сообщение и передается контроллеру), Flow-Removed сообщения (информирование контроллера об изменениях в таблице потоков коммутатора) и др. Среди сообщений контроллера коммутатору как наиболее часто отсылаемые можно выделить Packet-out и Flow-mod сообщения. Указанные сообщения посылаются в качестве ответа на Packet-in сообщение. Сообщение Packet-out указывает коммутатору отослать пакет потока на соответствующий порт коммутатора, сообщение Flow-mod указывает установить правило в таблицу потоков коммутатора. Симметричные сообщения включают в себя Hello сообщения, которыми коммутатор и контроллер обмениваются при установлении соединения, Echo сообщения и сообщения об ошибках.

Таким образом, контроллер подвергается большой нагрузке со стороны сетевых устройств: обработка входящих сообщений, формирование ответов для большого числа запросов на установление новых потоков. В связи с тем, что высокая нагрузка может отрицательно сказаться на производительности и стабильности работы контроллера (вплоть до отказа), целесообразно отслеживать параметры текущей загрузки контроллера, среди которых можно назвать использование физических ресурсов сервера, частоту получения Packet-in сообщений по сегменту сети, число потоков, поддерживаемых контроллером и др. Данные о загрузке контроллеров РСОС могут использоваться для перераспределения управления коммутаторами между ними(например, передать управление коммутатором на менее нагруженный контроллер распределенной платформы управления). Кроме нагрузки со стороны сетевых устройств, контроллер подвергается нагрузке со стороны приложений (ресурсоемкие

приложения могут замедлить работу контроллера), а также со стороны других контроллеров (поддержание общего представления сети, перераспределение коммутаторов между контроллерами), со стороны системных администраторов (изменение настроек функционирования, несанкционированный доступ). Перечень параметров функционирования контроллера, которые целесообразно отслеживать в рамках задачи мониторинга состояния контроллера распределенной платформы управления, представлены в [10].

2. Распределенная платформа управления на базе контроллера RUNOS

RUNOS [7], [8] – OpenFlow-контроллер, разработанный в Центре прикладных исследований компьютерных сетей (ЦПИКС), с открытым исходным кодом для использования в корпоративных сетях и сетях провайдеров. RUNOS реализован на языке C++ с использованием механизмов многопоточности, библиотека QT 5 и Boost.ASIO.

Возможный вариант построения распределенной платформы управления: контроллеры RUNOS, поддерживающие доступ к распределенному хранилищу данных (рис. 2). Рассмотрим программно-конфигурируемую сеть, коммутаторы которой некоторым образом разделены на сегменты. Коммутаторы каждого сегмента сети соединены с одним из контроллеров RUNOS. Сетевые приложения, запущенные на контроллерах, обмениваются друг с другом данными посредством распределенного хранилища данных, формируя, таким образом, общее представление сети (например, менеджер топологии, запущенный на одном контролере, может хранить граф своего сегмента сети в распределенном хранилище и, имея доступ к графам других сегментов, при необходимости строить граф всей сети). Описанную систему далее будем считать распределенной платформой управления на базе контроллеров RUNOS.

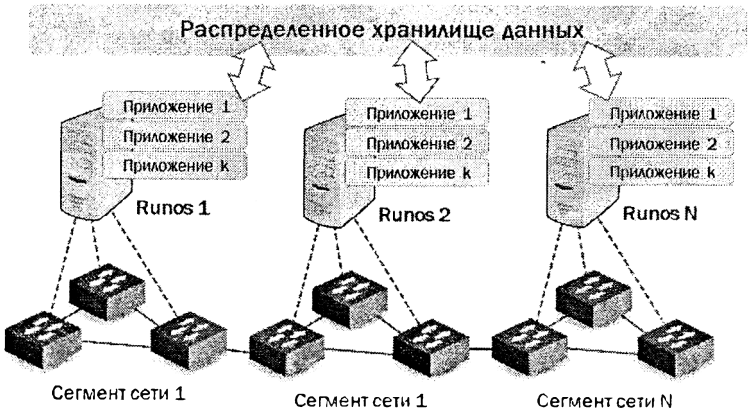

Рис. 2. Распределенная платформа управления на базе контроллера RUNOS

3. Архитектура приложения для мониторинга состояния контроллера

Механизм мониторинга состояния контроллера реализован как приложение Controller Monitor для контроллера RUNOS. Приложение осуществляет сбор данных о работе контроллера, фиксирует текущие значения параметров функционирования контроллера, которые заносятся в in-memory NoSQL базу данных Redis [9] (нерсляционная высокопроизводительная СУБД). Через заданные интервалы времени приложение обращается к in-memory базе данных, записывает в csv-файлы на жесткий диск данные по отдельным контроллерам и по платформе в целом, собранные за интервалы времени с момента последней записи.

Требования, предъявляемые к приложению Controller Monitor мониторинга состояния контроллера RUNOS, приведены в работе [10].

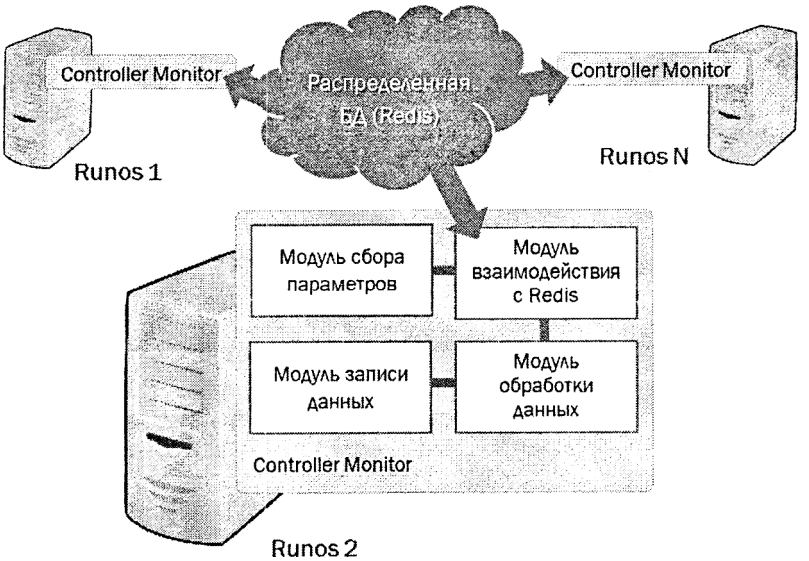

Рис. 3. Архитектура распределенной платформы управления с приложением Controller Monitor

Приложение мониторинга Controller Monitor состоит из четырех модулей (рис. 3).

Модуль взаимодействия с Redis. Взаимодействие с хранилищем данных Redis (работа с JSON-объектами и строками). Redis представляет собой хранилище типа «ключ-значение».

Модуль сбора данных. Собираемые приложением данные разбиты на группы.В конфигурационном файле приложения можно

задать, сбор каких групп характеристик будет осуществляться. Приложение Controller Monitor реализует мониторинг следующих характеристик (также указаны компоненты ключей, с которыми ассоциированы значения параметров в базе данных Redis):

- Статистика OpenFlow-сообщений: число поступивших Packet-in сообщений за заданный интервал времени (подсчет производится во время обработки входящего Packet-in сообщения); число отправленных Packet-out и Flow-mod сообщений за заданный интервал времени (при подсчете значения параметра используется API контроллера).
- Информация о потоках: общее число поддерживаемых контроллером RUNOS потоков в сегменте сети (проводится мониторинг изменения состояния потоков); число новых потоков в сегменте сети за заданный интервал времени (подсчет производится во время обработки входящего Packet-in сообщения); число устаревших потоков в сегменте сети за интервал времени (проводится мониторинг изменения состояния потоков).
- Информация о соединениях с коммутаторами: число коммутаторов в сегменте сети контроллера (подсчитывается как число активных соединений с коммутаторами, используется API контроллера); число потерянных соединений с коммутаторами за интервал времени (для мониторинга используется API контроллера); задержка на линии между контроллером и каждым коммутатором его сегмента сети (измеряется как время между отправкой Echo-Request сообщения коммутатору и получением Echo-Reply сообщения).
- Загрузка физических ресурсов серверов контроллеров (CPU, RAM): процент использования центрального процессора (использовалась библиотека StatGrab [10]); процент использования оперативной памяти (использовалась библиотека StatGrab [10]).

Мониторинг проводится независимо по всем указанным выше параметрам. При изменении значения какого-либо параметра происходит обращение к распределенной базе данных Redis по соответствующему ключу и изменение его значения.

Модуль обработки данных. Через заданные интервалы времени запускается модуль обработки данных, который извлекает из базы данных Redis доступную информацию по всем контроллерам. Фиксируется максимальное значение каждого параметра за заданный интервал времени. Данные передаются модулю записи данных.

Модуль записи данных. Реализуется запись собранных данных в csv-файлы: файл данных по отдельному контроллеру, файл для задержек между контроллерами и коммутаторами (по всей платформе), файл данных по всей платформе управления.

4. Экспериментальное исследование

Целью экспериментального исследования являлась проверка корректности функционирования разработанного приложения Controller Monitor мониторинга состояния контроллера RUNOS в составе распределенной платформы управления. Приложение считается корректно функционирующим, если оно позволяет собирать данные в соответствии с требованиями к приложению и при этом не создает существенных накладных расходов на их сбор. Методика исследования учитывает, что данные, собираемые приложением Controller Monitor, являются приближенными за заданный интервал времени и не могут использоваться в качестве точных характеристик состояния контроллера. Результаты работы могут быть использованы только для получения представления о состоянии распределенной платформы управления на базе RUNOS.

Экспериментальный стенд представляет собой распределенную платформу управления на базе RUNOS (см. рис. 2) из трех контроллеров, взаимодействующих с распределенной базой данных Redis, каждый из которых соединен с запущенной средой эмулирования сетей Mininet [11] (таким образом, каждый контроллер подключен к своему сегменту сети). Технически, в работе экспериментального стенда участвуют две машины, соединенные витой парой (Fast Ethernet) через интернет-маршрутизатор, используются IP-псевдонимы для интерфейсов. На машине 1 запущено три контроллера RUNOS с активными приложениями Controller Monitor и Learning Switch. На машине 2 работает база данных Redis, а также запущено три экземпляра среды эмулирования сетей Mininet с линейной топологией "linear,5,4" (по 4 хоста на каждый из 5 коммутаторов), каждый из которых соединен с определенным контроллером RUNOS на машине 1.

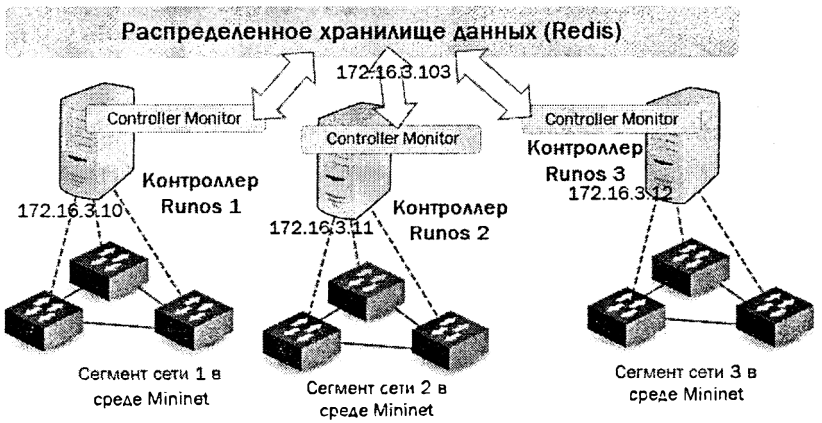

Рис. 4. Логическая схема экспериментального стенда

В рамках данной работы было проведено три эксперимента .

1. Проверка сбора статистики OpenFlow-сообщений, данных о потоках и данных об использовании физических ресурсов. Экспериментальный стенд во время проверки запускался дважды. Приложение мониторинга контроллера Controller Monitor запускалось со следующими параметрами:
осуществлсние сбора статистики OpenFlow-сообщений, потоков, использования физических ресурсов;
осуществление сбора данных по загрузке платформы в целом;
запись собранных данных по контроллеру с интервалом времени 3 с (1 с при втором запуске);
запись собранных данных по платформе с интервалом времени 3 с (2 с при втором запуске);
сбор данных по всей платформе осуществляется только контроллером RUNOS 1.

В ходе проверки сравнивались число Packet-in и сумма числа Packet-out и Flow-mod сообщений, а также проверялась корректность сбора данных по всей платформе управления.

2. Проверка сбора данных по задержке коммутатор-контроллер. Приложение Controller Monitor запускалось с парамстрами, аналогичными параметрам проверки 1, со следующими изменениями: сбор данных проводится только для режима измерения задержки коммутатор-контроллер.

В ходе проверки проверялись число соединений с коммутаторами и точность измерения задержек.

3. Определение нагрузки на контроллер, создаваемой приложением Controller Monitor. В качестве нагрузки на контроллер, создаваемой приложением, рассматривается разница во времени, затрачиваемом контроллером на обработку одного Packet-in сообщения с запущенным приложением и с неактивным приложением.

Приложение Controller Monitor запускалось с параметрами, аналогичными параметрам проверки 1, со следующими изменениями: проводится сбор всех доступных характеристик функционирования контроллера; сбор данных по всей платформе осуществляется на всех контроллерах.

В ходе проверки определялось и сравнивалось время, затрачиваемое контроллером на обработку одного Packet-in сообщения с активным и неактивным приложением Controller Monitor при помощи утилиты измерения производительности контроллера cbench.

Результаты экспериментального исследования приведены в таблицах 1–4 и на рис. 5.

Таблица 1

Результаты проверки сбора данных для платформы управления в целом. Интервал записи данных по контроллеру – 3 с, интервал записи данных по всей платформе управления – 3 с. Ошибка – среднее разницы между значениями параметров. Процент совпадений считается от числа записей в таблице выходного csv-файла

Сравниваемые значения параметров (сумма по всем контроллерам и значение по всей платформе)	Точных совпадений	Ошибка	Совпадений с учетом ошибки
Packet-in	92,31%	0,28 сообщения	92,31%
Packet-out + Flow-mod	70,88%	1,91 сообщений	84,62%
Общее число потоков	93,96%	0,16 потока	93,96%
Число новых потоков	88,46%	0,53 потока	88,46%
Число устаревших потоков	100,00%	0,01 потока	100,00%

Таблица 2

Результаты проверки сбора данных для всей платформы управления. Интервал записи данных по контроллеру – 1 с, интервал записи данных по всей платформе управления – 2 с

Сравниваемые значения параметров (сумма по всем контроллерам и значение по всей платформе)	Точных совпадений	Ошибка	Совпадений с учетом ошибки
Packet-in	72,74%	4,78 сообщений	85,22%
Packet-out + Flow-mod	74,38%	4,45 сообщений	85,06%
Общее число потоков	36,75%	10,50 потоков	79,01%
Число новых потоков	49,92%	11,11 потоков	82,76%
Число устаревших потоков	99,84%	0,1 потока	99,84%

Таблица 3

Результаты измерения задержек между каждым контроллером и каждым коммутатором. Средняя задержка – средняя задержка за время работы экспериментального стенда. Минимальная задержка – минимальная задержка за время работы стенда. Столбец «Коммутатор<i>:<j>» показывает значение задержки между коммутатором j и контроллером i

Пара-метр	Коммутатор 1:1	Коммутатор 1:2	Коммутатор 1:3	Коммутатор 1:4	Коммутатор 1:5	Коммутатор 2:1	Коммутатор 2:2	Коммутатор 2:3	Коммутатор 2:4	Коммутатор 2:5	Коммутатор 3:1	Коммутатор 3:2	Коммутатор 3:3	Коммутатор 3:4	Коммутатор 3:5
Сред. задержка, мс	5,9	5,0	5,8	5,5	6,1	5,8	5,3	6,0	5,5	4,9	8,7	8,2	8,8	8,5	8,0
Мин. задержка, мс	3,8	3,7	3,4	3,4	3,9	4,0	3,6	3,9	3,7	3,7	3,8	3,4	3,7	3,5	3,7

Таблица 4

Результаты тестирования контроллеров распределенной платформы управления на базе RUNOS средством измерения производительности cbench в рамках проверки 3

Конфигурация контроллера RUNOS	Мин. время обработки сообщ., мс	Макс. время обработки сообщ., мс	Сред. время обработки сообщ., мс
Контроллер с активным приложением Controller Monitor	0,064	0,107	0,084
Контроллер с неактивным приложением Controller Monitor	0,060	0,090	0,074
Добавочная нагрузка, % от нагрузки с неактивным приложением	6,38%	19,42%	14,14%

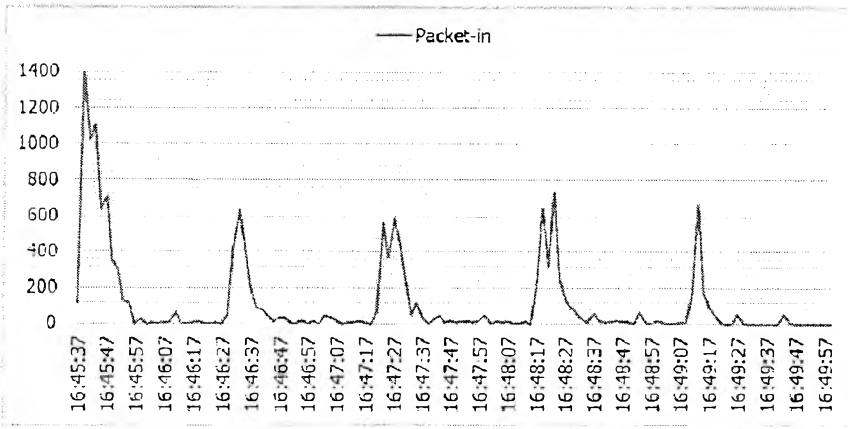

Рис. 5. Динамика изменения количества Packet-in сообщений, поступающих на контроллер RUNOS, для сети с топологией Mininet "linear,5,4", построенная на основе данных, собранных приложением Controller Monitor. Трафик генерировался утилитой ping. 16:45:37 – время первого запуска утилиты ping. Скачкообразность графика объясняется устареванием и удалением правил на коммутаторах, в связи с чем коммутатор, получив пакет с неизвестным ему адресатом, отправляет его на контроллер

По итогам проведенного экспериментального исследования можно сделать следующие выводы:

Для получения более точных значений параметров, измеряемых приложением Controller Monitor, необходимо задать меньший интервал времени для фиксации данных в файл, чем установленный по умолчанию (например, 1 с вместо 3 с).

Статистика для распределенной платформы управления на базе RUNOS, собираемая разработанным приложением, допускает бóльшую погрешность, чем статистика для отдельного контроллера.

Средняя загрузка контроллера, которую создает разработанное приложение Controller Monitor составляет в среднем порядка 15%, что позволяет использовать приложение для мониторинга подвергающихся большой нагрузке распределенных платформ управления.

Заключение

В рамках работы было разработано приложение Controller Monitor для контроллера RUNOS, осуществляющее мониторинг состояния контроллера в рамках распределенной платформы управления ПКС в реальном времени, и проведены экспериментальные исследования разработанного приложения.

В качестве возможного направления дальнейших исследований можно указать исследование методик обнаружения угроз (отказов, перегрузок, атак) и их прогнозирования на основе собранной информации о состоянии контроллеров распределенной платформы управления.

Литература

1. Смелянский Р.Л. Программно-конфигурируемые сети // Открытые системы. №9. 2012. С. 15-26.
2. Open Networking Foundation. Software-Defined Networking: The New Norm for Networks // ONF White Paper, 2012.
3. Open Networking Lab. Introducing ONOS – a SDN network operating system for Service Providers // ONOS Whitepaper, 2014.
4. Pashkov V., Shalimov A., Smeliansky R. Controller Failover for Enterprise SDN // In Proceedings of the Modern Networking Technologies (MoNeTec'2014). IEEE, 2014. P. 27-32.
5. ODL Beryllium (Be) – The Fourth Release of OpenDaylight: https://www.opendaylight.org/odlbe
6. Open Networking Foundation. OpenFlow Switch Specification Version 1.3.0 (Protocol version 0x04) // ONF, 2012.
7. Shalimov A. et al. The RUNOS OpenFlow Controller //2015 Fourth European Workshop on Software Defined Networks (EWSDN). – IEEE, 2015. – С. 103-104.
8. Контроллер RUNOS: https://github.com/ARCCN/RUNOS
9. Redis: http://redis.io/.
10. Скобцова Ю.А. Исследование и разработка сетевого приложения для распределенной платформы управления в программно-конфигурируемых сетях – Курсовая работа //Московский государственный университет имени М.В. Ломоносова — 2016. [PDF] (https://cloud.lvk.cs.msu.su/index.php/s/l1h1XPol39Gzw9c).
11. StatGrab – libstatgrab homepage: https://www.i-scream.org/libstatgrab/
12. Mininet Overview – Mininet: http://mininet.org/overview/.

Сычева Е.А., Пашков В.Н.

Исследование и разработка алгоритмов обнаружения и предотвращения DDoS атак на контроллер в программно-конфигурируемых сетях[1]

Введение

Распределенные атаки типа «отказ в обслуживание» (Distributed Denial of Service, DDoS) получили широкое распространение в компьютерных сетях в связи с простотой реализации и доступностью инструментов для поиска уязвимостей и анализа на этапе подготовки атаки как один из надежных способов гарантированного достижения целей [1]. Стоит заметить, что в качестве устройства, на которое нацелена атака, может выступать не только хост-жертва, но и сетевое оборудование. Однако, если в рамках традиционной архитектуры компьютерных сетей данный тип атаки является узкоспециализированным и потому не так распространен, то программно-конфигурируемые сети (ПКС, Software-Defined Networks, SDN), предлагающие перейти от управления отдельными сетевыми устройствами к управлению сетью в целом [2], предоставляют злоумышленникам большой простор.

Управление сетью в ПКС логически централизованно. С одной стороны, это упрощает задачу управления сетью, позволяя целиком сосредоточить функции управления сетевыми устройствами и потоками данных в сетевых приложениях для контроллера. С другой стороны, это неизбежно ведет к появлению узкого места в сети — контроллера. Контроллер представляет собой специальное программное обеспечение, работающее на некотором сервере. Адрес и порт, по которому следует искать контроллер, известен администратору сети и указывается им в конфигурации каждого коммутатора. Отказ контроллера или сервера, либо изменение его адреса, может привести к сбою в работе всего управляемого им сетевого оборудования и, следовательно, к отказу всей сети.

Кроме того, к эффектам DDoS атак на контроллер можно отнести повышенные задержки и потерю пакетов для всей сети; ухудшение работы сетевых протоколов на коммутаторах — наиболее уязвимы протоколы, требующие постоянного общения с контроллером.

---

[1] Проект поддержан грантом Инновационного фонда «Сколково» от 2 июля 2012г. №79.

Главное отличие DDoS атак на контроллер в ПКС от DDoS атак на хост заключается в том, что атака, выполняемая в одном месте в сети, может повлиять на глобальное поведение сети.

1. Анализ DDoS атак на контроллер в ПКС

Условием реализации распределенных атак, направленных на отказ в обслуживании контроллера в ПКС, является возможность получения им *асинхронных сообщений* — коммутатор посылает данные, не дожидаясь запроса от контроллера — по внешнему событию. В случае протокола OpenFlow, наиболее перспективного и активно развивающегося стандарта для ПКС [2], таким внешним событием является получение пакета, для которого не найдено правило в таблицах коммутатора.

Одной из особенностей протокола OpenFlow является то, что существует два способа установки правил в таблицах коммутаторов (при помощи другой группы сообщений: *сообщений, отправляемых контроллером коммутатору*): проактивно и реактивно. В первом случае контроллер самостоятельно принимает решение об установке того или иного правила, не дожидаясь запроса от коммутатора. Во втором установка правил инициируется асинхронным запросом от коммутатора, что обуславливает риск атак, направленных на отказ в обслуживании контроллера.

Механизм обработки *нового запроса*— признака нового потока — можно представить, как следующую последовательность шагов:

1. Коммутатор получает пакет, не удовлетворяющий ни одному из присутствующих на данный момент в таблице потоков, и применяет специальное правило по умолчанию (*table miss*) [3] — отправляет запрос контроллеру на установление нового потока как асинхронное Packet-In сообщение.

2. Контроллер анализирует поля заголовков пакета и устанавливает на коммутаторе правило в соответствии с некоторым алгоритмом — определяется работающими на контроллере приложениями, — при необходимости вычисляя путь маршрутизации для нового потока.

Установка нового правила на коммутаторе также называется установкой нового потока, так как данное правило определяет процесс обработки некоторого множества пакетов, значения заголовков которых удовлетворяют ему.

Большое количество новых запросов на установление новых потоков, полученное в течение некоторого малого промежутка времени, способно «насытить» реактивно устанавливающий правила контроллер и, тем самым, вызвать отказ в обслуживании легитимных пользователей. Таким образом, начало DDoS атаки может быть быстро обнаружено на

ранних этапах по высокой частоте получения запросов на установления новых потоков в сети.

2. Постановка задачи

Пусть дана программно-конфигурируемая сеть, управляемая контроллером, реактивно устанавливающим правила. Взаимодействие между сетевыми устройствами и контроллером осуществляется по протоколу OpenFlow.

Сеть должна иметь механизмы обнаружения и предотвращения DDoS-атак на контроллер. Предполагается, что:

- Злоумышленник знает о том, что сеть является программно-конфигурируемой. Он может определить это, проверяя существует ли разница между временем ответа первого пакета и временем ответа последующих пакетов.
- Злоумышленник контролирует некоторое множество хостов в сети (далее — нелегитимные пользователи сети), которые используются для отправки специально обработанных пакетов. Нелегитимных пользователей, используемых в рамках данной модели угроз, можно разделить на два типа:
  - Злонамеренный пользователь посылает пакеты, адрес получателя в которых генерируется случайным образом.
  - DDoS пользователь посылает пакеты, адреса отправителя и получателя в которых генерируются случайным образом.

Хосты, не контролируемые злоумышленником, будем называть *легитимными пользователями* сети.

Модель угрозы DDoS атак на контроллер в ПКС формулируется следующим образом:

- Нелегитимные пользователи посылают большое количество *аномальных пакетов* -- специально обработанных пакетов, некоторое непустое подмножество полей заголовка которых подменено случайными величинами.
- Поскольку аномальный пакет с большой вероятностью не соответствует правилам в таблице потоков, по умолчанию коммутатор отправляет контроллеру его заголовок в составе Packet-In сообщения.
- Контроллер формирует соответствующее правило для аномальных пакетов данного типа и устанавливает его на коммутаторе (или на наборе управляемых им коммутаторов) [2].
- По прошествии некоторого времени контроллер будет сбрасывать пакеты из-за высокой нагрузки, что свидетельствует об успехе DDoS атаки. Следует отметить, что повышение нагрузки обуславливается не только процедурой обработки Packet-In сообщений, но и такими факторами, как загрузка процессора по причине поддержки множества защищенных соединений.

3. Методы обнаружения и предотвращения DDoS атак на контроллер в ПКС

В [9] приведен обзор существующих методов обнаружения DDoS атак на контроллер (на основе мониторинга загрузки контроллера, на основе мониторинга валидности адресов получателей). К сожалению, на сегодняшний день, ни один из них не учитывает в достаточной мере специфику атаки, а именно отсутствие выделенного целевого хоста, и при этом экспериментально подтверждает свою состоятельность. Помимо того, в методе на основе мониторинга валидности адресов получателей не поднимается вопрос обнаружения окончания DDoS атаки на контроллер, который важен с точки зрения отключения механизмов предотвращения DDoS атак на контроллер, что соответствует переходу системы в нормальное состояние.

Существует несколько подходов к решению задачи предотвращения DDoS атак на контроллер в ПКС (миграция соединения, замена дисциплины обслуживания Packet-In запросов в буфере контроллера, классификация пользователей, установка проактивных правил в таблицах потоков). Обзор существующих методов решения данной задачи приведен в [9]. По критерию оптимальности — минимизации следующих метрик:

- средняя задержка установления потока для легитимных пользователей: в нормальных условиях и под DDoS атакой на контроллер;
- накладные расходы по производительности контроллера;

выбран статистический метод на основе классификации пользователей [4] для решения поставленной в разделе 2 задачи.

В основу метода легли следующие статистические факты:

- Около 90% «частых» — что, в данном контексте, эквивалентно легитимным — пользователей отправляют, по крайней мере, 5 пакетов в каждый пункт назначения. И наоборот, ненормальные пользователи передают менее 5 пакетов на соединение (ошибочные пакеты или пакеты DDoS атаки).
- Около 60% IP-адресов возникают только один день в течение двух недель. Большинство из них являются «нечастыми» пользователями или адресами, использующимися в ходе DDoS атаки.

Исходя из этого, в качестве параметров настройки алгоритма используются:

- минимальное количество пакетов на соединение «частого» пользователя — n;
- среднее количество соединений, которые устанавливают «частые» пользователи — k.

На стороне контроллера объявляется временная таблица T, которая используется для хранения IP-адреса отправителя пересылаемых коммутатором пакетов, а также соответствующего ему счетчика поступивших пакетов c_i.

Во время атаки отправитель нового пакета, то есть пересылаемого на контроллер, рассматривается, в первую очередь, как злоумышленник. Контроллер создаст новую запись в таблице правил коммутатора с короткими таймерами: максимальным временем жизни и максимальным временем простоя, значения которых меньше, чем соответствующие значения для нормальных записей. Кроме того, счетчик c_i соответствующего IP-адреса увеличивается на 1.

Когда c_i достигает k, с коммутатора запрашивается счетчик среднего числа пакетов s. То, что s больше n, трактуется, как доказательство легитимности пользователя. Следовательно, контроллер сбрасывает таймеры существующих в таблице правил коммутатора записей к значениям по умолчанию, соответствующим нормальному режиму работы. И наоборот, если s меньше n, трафик классифицируется как вредоносный, и контроллер устанавливает правило сброса на коммутаторы.

Таким образом, описанный метод обеспечивает:

- сброс пакетов злоумышленника на стороне коммутатора;
- обслуживание пакетов источников, использующихся в ходе DDoS атаки, с меньшими значениями таймаутов;
- обслуживание «частых» пользователей со значениями таймаутов по умолчанию.

4. Разработка методов обнаружения и предотвращения DDoS атак на контроллер в ПКС

Проанализируем выбранный за основу метод предотвращения DDoS атак на контроллер: статистический метод на основе классификации пользователей. В таблица 1 представлен заголовок таблицы пользователей T.

Таблица 1

Заголовок таблицы пользователей T

IP-адрес отправителя	Счетчик соединений, c_i

Достоинством данной табличной схемы хранения является ее компактность. К основным ее недостаткам можно отнести:

- отсутствие какой-либо информации о легитимности пользователя-отправителя, что фактически означает отсутствие какой-либо разницы между новым потоком легитимного и нелегитимного пользователей;

- отсутствие ограничений размера таблицы, а также какой-либо процедуры удаления части записей.

Для иллюстрации важности вышеназванных недостатков воспользуемся таблицей 2.

Таблица 2

Хранение данных о различных типах пользователей

Тип пользователей	Количество записей в таблице T	Счетчик соединений, c_i	Счетчик пакетов, s_i
Легитимный пользователь	1	$c_i \geq k$	$s_i > n$
		$c_i < k$	
Нелегитимный злонамеренный пользователь	1	$c_i \geq k$	$s_i \leq n$
Нелегитимный DDoS пользователь	l	$c_{i..i+l} \leq k^{DDoS}$	$s_{i..i+l} \leq n$

Во-первых, до тех пор, пока $c_i < k$, легитимный пользователь рассматривается как злоумышленник, и его пакетам ставятся в соответствие правила потоков с короткими таймерами. Это искусственно увеличивает нагрузку на контроллер, а также должно учитываться при использовании подобного трафика для расчета k.

Во-вторых, предположим, что одному нелегитимному DDoS пользователю соответствует l записей в таблице T. Поскольку для осуществления DDoS атак используется большое количество компьютеров, без ограничений подобная таблица за короткое время приведет к замедлению работы приложения, а, следовательно, и увеличению средней задержки установления потока для легитимных пользователей.

Помимо того, отмечены еще два важных недостатка метода:

- Постоянная работа активных контрмер против DDoS атак на контроллер негативно влияет на его производительность.
- Параметры метода k и n фиксированы, что не дает возможности своевременно реагировать на изменение поведения пользователей.

Рассмотрим, как можно модифицировать метод, чтобы преодолеть обнаруженные недостатки.

Во-первых, заменим постоянный режим работы активных контрмер на режим с выделенными стадиями обнаружения начала и завершения DDoS атаки на контроллера, которые будут соответственно

включать и выключать механизмы противодействия. Для реализации стадий обнаружения введем процедуры постоянного мониторинга количества записей и скорости изменения таблицы T. Под *скоростью изменения таблицы* T будем понимать количество операций добавления для нелегитимных DDoS пользователей и изменение записей в связи с классификацией — для нелегитимных злонамеренных.

В связи с добавлением подобного разделения возникает необходимость разработки формального критерия принадлежности пользователя к тому или иному нелегитимному типу. Для этого введем еще один параметр метода k^{DDoS} — максимальное число соединений нелегитимного DDoS пользователей. Таким образом, счетчики соединений DDoS пользователя $c_{i..i+l} \le k^{DDoS}$.

Параметрами процедуры постоянного мониторинга являются:

- пороговые значения количества записей, соответствующих злонамеренным и DDoS пользователям;
- окно времени — t;
- пороговые значения количества изменений записей, соответствующих злонамеренным и DDoS пользователям за t.

Во-вторых, вместо фиксации параметров разрешим их динамическую коррекцию при условии фиксации верхнего и нижнего пороговых значений. Последнее особенно важно для избегания злонамеренного обучения алгоритма. Обозначения верхних и нижних пороговых значений параметров метода: $n^{\min} \le n \le n^{\max}$ и $k^{\min} \le k \le k^{\max}$.

Помимо усреднения числа соединений, устанавливаемых всеми легитимными пользователями, введем k_i — среднее число соединений, устанавливаемых легитимным пользователем i. Это повысит быстроту реагирования на изменение поведения легитимных пользователей, например, на захват их злоумышленником.

В-третьих, разделим таблицу T на две:

- таблица легитимных пользователей T_v (см. таблица 3)
- таблица нелегитимных пользователей T_{nv}.

Таблица 3

Заголовок таблицы легитимных пользователей T_v

IP-адрес отправителя	Счетчик соединений, c_i	Среднее число соединений, k_i

Таблица T_v формально не ограничена, поскольку содержит сведения о легитимных пользователях — наиболее ценные, с точки зрения обеспечения качества, данные. Для ограничения размера таблицы T_{nv} добавим фиксированные для злонамеренных и DDoS пользователей таймеры записей, аналогичные таймерам в таблице правил коммутатора (см. таблица 4).

Таблица 4

Заголовок таблицы нелегитимных пользователей T_{nv}

IP-адрес отправителя	Счетчик соединений, c_i	Таймеры	
		Максимальное время жизни, t_i^h	Максимальное время простоя, t_i^i

Кроме того, введем процедуру периодического удаления части записей таблицы T_{nv}, задающуюся следующими параметрами:

- период — t';
- максимальный размер таблицы T_{nv};
- пороговое значение количества записей таблицы T_{nv}.

Организация хранения статистики легитимных и нелегитимных пользователей отдельно друг от друга также решает проблему отсутствия информации о легитимности пользователя.

Последнее предлагаемое изменение касается получения статистики с коммутатора: вместо того, чтобы запрашивать счетчик среднего числа пакетов s_i, запросим счетчики всех m правил потоков $S_i^1..S_i^m$, для которых адрес проверяемого пользователя соответствует маске адреса источника. Для подтверждения или опровержения легитимности пользователя воспользуемся максимально допустимым процентом невалидных потоков p.

5. Экспериментальное исследование

Для экспериментальных исследований использовались:

- система прототипирования компьютерной сети *Mininet* [6];
- OpenFlow-контроллер *RUNOS* [7], для которого было разработано приложение, реализующее предложенные в главе 4 алгоритмы;
- библиотека *Scapy-Python* [8] для генерации специально сконструированных сетевых пакетов.

Топология эксперимента состоит из четырех хостов: сервера и трех пользователей различных типов (легитимного, нелегитимного

злонамеренного и нелегитимного DDoS пользователя, соответственно), OpenFlow-коммутатора и OpenFlow-контроллера (рис. 1).

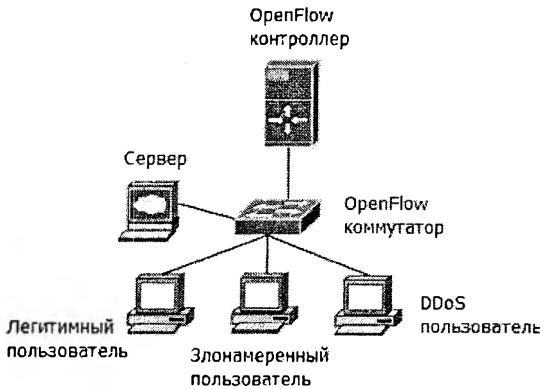

Рис. 1. Логическая схема экспериментального стенда

При этом:

- Легитимный пользователь устанавливает 5 различных соединений с сервером, в рамках которых передает от 5 до 10 пакетов на соединение.
- Злонамеренный пользователь бесконечно посылает специально сконструированные пакеты, адрес получателя в которых генерируется случайным образом.
- DDoS пользователь бесконечно посылает специально сконструированные пакеты, адреса отправителя и получателя в которых генерируется случайным образом.

Экспериментальное исследование показало, что скорость обнаружения DDoS атак на контроллер зависит от типа нелегитимного трафика:

- Если атака осуществляется только нелегитимными злонамеренными пользователями, то момент обнаружения предсказуем и совпадает с моментом времени, соответствующим первому подтверждению нелегитимности пользователя, полученного на основе запрошенных с коммутатора счетчиков числа пакетов (количественная оценка в терминах новых пакетов этому моменту времени дается далее).
- Поскольку атака нелегитимными DDoS пользователями обнаруживается по средствам некоторого множества критериев, каждый из которых имеет свой весовой коэффициент, важными становятся количественные значения ограничений критериев и их весовых коэффициентов.
- В случае атаки различными типами нелегитимных пользователей эти факторы накладываются, увеличивая скорость обнаружения.

Приведем результаты, полученные при исследовании алгоритмов. Фиксированные параметры приложения:

- значения коротких таймеров в таблице правил коммутатора: $t^h = 60$ и $t^i = 10$ секунд;
- $k^{DDoS} = 2$ соединения; $t_i^h = 6000$, $t_i^i = 600$ секунд;
- $n^{\min} = 3$, $n^{\max} = 7$ пакетов; $k^{\min} = 3$, $k^{\max} = 7$ соединений;
- период запуска процедуры обновления среднего числа соединений, устанавливаемых легитимными пользователями — 100 секунд; процедуры обнаружения — 5 секунд; процедуры удаления части записей таблицы нелегитимных пользователей — 500 секунд.

Основные результаты исследования метода предотвращения DDoS атак на контроллер:

- Пакеты нелегитимного злонамеренного пользователя сбрасываются на уровне передачи данных: перед установкой правила сброса после обработки k соединений.
- Пакеты нелегитимного DDoS пользователя обслуживаются правилами потоков с короткими таймерами t^h и t^i.
- Пакеты легитимного пользователя обслуживаются правилами потоков со значениями таймеров по умолчанию (например, $t^h = 300$ и $t^i = 60$ секунд).

Заключение

В рамках данной работы были разработаны, реализованы и исследованы алгоритмы обнаружения и предотвращения DDoS атак на контроллер в программно-конфигурируемых сетях.

Возможным направлением дальнейших исследований является исследование применимости предложенных алгоритмов для защиты распределенной платформы управления в ПКС.

Литература

1. Терновой О.С., Шатохин А.С. Снижение ошибки обнаружения DDOS атак статистическими методами при учете сезонности. // Ползуновский вестник. 2012.

2. Смелянский Р.Л. Программно-конфигурируемые сети // Открытые системы [Электронный ресурс]. — URL: http://www.osp.ru/os/2012/09/13032491 (дата обращения: 15.06.2016).

3. Чемерицкий Е.В. Исследование методов контроля функционирования программно-конфигурируемых сетей: дис. канд. физ.-мат. наук: 05.13.11. Москва, 2015.

4. Dao, N.N., Park J., Park M., Cho S. A feasible method to combat against DDoS attack in SDN network // Information Networking (ICOIN), 2015 International Conference on. 2015. P. 309-311.

5. Gde Dharma N.I., Muthohar M.F., Prayuda J.D., Priagung K., Choi D. Time-based DDoS detection and mitigation for SDN controller. // Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific. 2015. P. 550-553.

6. Mininet homepage [HTML] (http://mininet.org/).

7. RUNOS homepage [HTML] (http://arccn.github.io/runos/).

8. Python-Scapy Module main site [HTML] (http://www.secdev.org/projects/ scapy/).

9. Сычева Е.А. Исследование и разработка алгоритмов обнаружения и предотвращения DDoS атак на контроллер в программно-конфигурируемых сетях 2016 Курсовая работа //Московский государственный университет имени М.В. Ломоносова, 2016. [PDF] (https://cloud.lvk.cs.msu.su/index.php/s/aZWPvcLepmLmvma).

Хахалин А.С., Чемерицкий Е.В.

Разработка метода организации централизованного контроля над сетью программно-конфигурируемых коммутаторов без использования вспомогательной сети управления[1]

Введение

Объектом рассмотрения настоящей работы являются *Программно-Конфигурируемые Сети* (ПКС) [1] – особый класс компьютерных сетей, в основу которого положена идея разделения сети на два контура: *контур данных*, отвечающий за непосредственную передачу пакетов между абонентскими машинами сети, и *контур управления*, который загружает на сетевое оборудование надлежащие настройки коммутации и маршрутизации. Для управления указанными настройками используется *контроллер* – единая программа, командам которой подчиняются все коммутационные устройства сети. Совместимость между контроллером и коммутаторами различных производителей достигается с помощью использования специализированных протоколов управления, которые позволяющих абстрагироваться от внутреннего устройства коммутатора. Наиболее распространённым из них является протокол OpenFlow [2].

Нередко для обеспечения взаимодействия между контроллером и коммутаторами используется дополнительное оборудование и выделенные линии связи. Поэтому сетевую инфраструктуру ПКС принято разделять на две части, каждая из которых поддерживает работу своего контура: *сеть контура данных* и *сеть контура управления*. Если сеть контура управления независима от сети контура данных, говорят, что говорят, что контроллер ПКС реализует *внешнее* (out-of-band) управление коммутаторами. Иначе взаимодействие между контроллером и коммутаторами обеспечивается с использованием тех же линий связи, по которым передаются пользовательские данные – такое управление принято называть *интегрированным* (in-band) [3].

Поскольку внешнее управление обеспечивает изоляцию служебного трафика от пользовательских данных, то оно является более удобным и безопасным. В тоже время внешнее управление требует прокладки дополнительных линий связи, поэтому его реализация зачастую слишком дорога и не осуществима в сетях, которые связывают объекты, находящиеся на значительном расстоянии друг от друга. В

[1] Работа выполнена при финансовой поддержке РФФИ, грант № 14-07-00625/16.

таких сетях единственно возможным является применение интегрированного управления, которое обладает одним существенным недостатком – высокой сложностью обеспечения надёжных соединений с контроллером. При таком управлении выход из строя любой линии связи приводит к разрыву связи контроллера со всеми коммутаторами, которые подключены через эту линию, и может вызвать перебои в работе целых сегментами сети, что недопустимо. Поэтому контроллеру необходимо заранее подготовить коммутаторы, заложив на них правила по перестроению маршрута передачи данных на случай каждого из возможных отказов.

Нетрудно видеть, что размер множества запасных маршрутов, необходимых для сохранения связи с коммутаторами при возникновении нескольких отказов оборудования, экспоненциально зависит от их количества. В достаточно больших сетях обеспечение устойчивости к нескольким отказам может потребовать от контроллера загрузить на коммутаторы десятки и сотни дополнительных правил по обработке пакетов. Поскольку современные коммутаторы зачастую поддерживают лишь несколько тысяч правил, то обеспечение высокого уровня устойчивости к отказам может привести к нехватке места на правила обслуживания пользовательского трафика. Поэтому разработчики контроллеров вынуждены идти на компромисс между надёжностью соединения и накладными расходами, связанными с её обеспечением.[2]

Настоящая работа предлагает новый метод построения надёжных соединений, который позволяет восстанавливать соединение между коммутатором и контроллером пока топология сети сохраняет хотя бы один путь между ними вне зависимости от количества произошедших отказов.

В разделе 2 рассматривается некоторые распространённые методы обеспечения отказоустойчивости соединений в ПКС. Основные принципы предложенного метода обеспечения отказоустойчивости изложены в разделе 3. Детальное описание реализации предложенного метода в терминах протокола OpenFlow содержится в разделе 4.

2. Существующие методы решения задачи

Самым распространённым методом защиты связи с контроллером является *статическое резервирование маршрутов* [7],[5]. Данный метод основан на предварительном расчете нескольких

[2] На практике при установке соединения между контроллером и коммутатором, как правило, делается предположение о том, что в сети не произойдёт более одного одновременного отказа, и после первого отказа контроллер успеет загрузить новый набор правила, который подготовят сеть к возникновению нового отказа.

маршрутов. Изначально для связи коммутатора с контроллером используется наилучший из них. В случае если контур данных детектирует отказ этого маршрута, он автоматически переключается на следующий по оптимальности. Таким образом, сеть сохранит связь с контроллером, если возникшие в ней отказы не повлияют на работоспособность хотя бы одного из заложенных в неё статических маршрутов.

Защита соединения на основе протокола OpenFlow реализуется за счет групповой таблицы. Групповая таблица может содержать, в частности, записи вида fast-failover, которые позволяют изменять правила обслуживания пакетов в зависимости от работоспособности подключённых к коммутатору линий связи. Каждая из таких записей представляет собой список пар из порта и набора действий. При обработке пакета коммутатор поочерёдно просматривает список в поисках первой пары, по порту которой есть связь. Если такая пара найдена, то к пакету применяются ассоциированные с ней действия, и обработка пакета завершается. Таким образом, установив соответствующие записи в групповую таблицу коммутатора, контроллер может обеспечить переключение коммутаторов на запасные маршруты при отказе линий связи, входящих в основной маршрут.

Методы статического резервирования маршрутов на практике часто не покрывают все возможные комбинации отказов. Как следствие, они не могут гарантировать работоспособность всех коммутаторов сети после возникновения отказов. Если же генерировать такой набор маршрутов, который покрывает все возможные варианты отказов, то количество правил, которые придется установить на коммутаторы, будет неприемлемо велико. Рассмотрим два варианта генерации маршрутов для покрытия всех возможных вариантов отказов.

Маршруты без возврата. Для обеспечения устойчивости к отказу произвольной линии связи основного маршрута без возврата трафика, нам нужно проложить L статических маршрутов, где L – число линий связи в этом маршруте. Тогда общее число маршрутов будет exp(L), а количество правил на коммутаторе N * exp(L).

Маршруты с возвратом. В случае обнаружение отказа линии связи на пути следования основного маршрута, происходит возврат на начальную вершину, и используется один из запасных маршрут. Таким образом, для каждого коммутатора требуется просчитать до (N – 1)! маршрутов, где N – общее количество коммутаторов в сети. Поскольку для каждого из указанных маршрутов необходимо установить правило на каждом коммутаторе вдоль этого маршрута, то количество правил на коммутаторе можно оценить величиной $o(\exp(N))$.

В работе предлагается оригинальный метод обеспечения отказоустойчивости, который позволяет обеспечить работу сети при наличии хотя бы одного маршрута между коммутаторами. При этом

количество правил, которые необходимо установить на коммутаторы меньше, чем при статическом резервирование маршрутов.

3. Предложенный метод

Сначала опишем идею разработанного алгоритма на конкретном примере (рис. 1). Если все линии связи сети с интегрированным управлением работоспособны, то для доставки пакетов от произвольного коммутатора к контроллеру достаточно обеспечить их передачу на любой из коммутаторов, соединённый с контроллером цепочкой из коммутаторов меньшей длины.

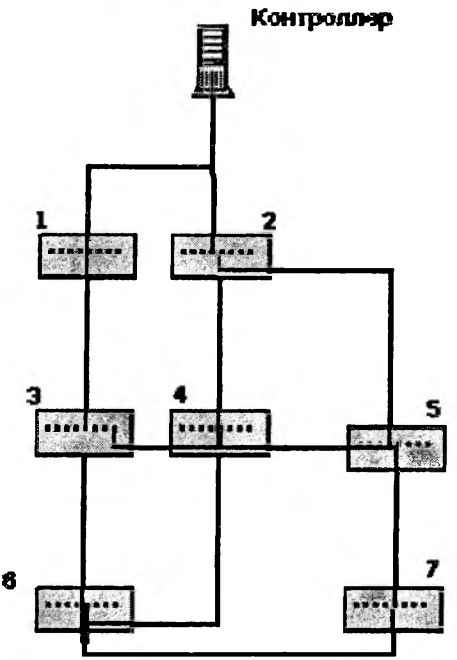

Рис. 1. Топология сети

Будем считать, что все вершины графа топологии сети по увеличению длины пути из этих вершин до вершины контроллера. Тогда при выборе следующей вершины для отправки пакета, следует выбирать вершину с наименьшим среди соседних вершин номером. Так для коммутатора 6 путь будет 6-3-1-0, где 0 - контроллер.

Если линия связи с такой вершиной разорвана, то выбранный путь передачи пакетов не может быть использован, и коммутатор пытается передать данные через соседнюю вершину с большим номером. Так, например, если будет разорвана линия связи 3-1, то путь для коммутатора 6 будет 6-3-4-2-0.

Если все линии связи кроме той, по которой пришел пакет, разорваны, то ни один путь, проходящий через эту вершину, не работает, и коммутатору необходимо отправить пакет обратно. Обратимся к рисунку 1. Если у коммутатора 3 произошел разрыв линий связи 3-1 и 3-4, то от коммутатора 3 не существует пути до контроллера. В этом случае пакет возвращается обратно, то есть на коммутатор 6. На коммутаторе 6 вершина 3 будет исключена из выбора, в виду отсутствия пути из этой вершины до контроллера. Тогда путь для коммутатора 6 будет следующий 6-4-2-0.

В какой-то момент пакет может попасть в вершину, где он уже был. Чтобы не допустить создания подобных циклов, необходимо знать, в каких вершинах уже побывал пакет. Для этого введем понятие *истории* – последовательности номеров, начинающихся номером коммутатора-источника и содержащих начальную часть пути, которым сеть пытается доставлять пакеты от этого коммутатора к контроллеру.

В общем случае контроллер должен заложить на каждый коммутатор такие правила, чтобы поступающие на его интерфейсы пакеты обрабатывались по следующему алгоритму:

- Отправить пакет на первую активную смежную вершину с наименьшим номером;
- При отправке пакета номер текущей вершины сохранить в историю;
- Если при отправке происходит возврат по истории, то текущую вершину не требуется писать в историю;
- Если текущая вершина есть в истории, и она является последней, то отправить на первую по возрастанию номеров активную смежную вершину, чей номер больше чем номер вершины, из которой пришел пакет;
- Если текущая вершина есть в истории, и она не является последней, то отправить на последнюю вершину в истории (иначе говоря, на порт, на который пришел пакет).

В следующем разделе приводится детальное описание того, как указанный алгоритм может быть закодирован правилами протокола OpenFlow.

4. OpenFlow реализация

В терминах OpenFlow логика работы коммутатора задаётся *таблицей правил* обработки пакетов. Каждое правило содержит набор простых *действий*, например, переписывание заголовка пакета или передача пакета через указанный порт, и *шаблона* – правило применяется к пакету лишь в том случае, если его заголовок подпадает под шаблон этого правила. Каждый бит шаблона может иметь одно из 3 значений: 0, 1 и *подстановочное* (*). Заголовок пакета попадает под шаблон лишь в том случае, когда каждый бит заголовка в точности равен соответствующему биту шаблона, или же в этом бите шаблона используется (*).

Для корректной работы программы коммутаторы в моделируемой сети должны поддерживать протокол OpenFlow не ниже версии 1.1, поддерживать группы типа fast-failover и иметь не менее 3 таблиц правил. Каждый пакет имеет заголовок, состоящий из нескольких полей. Каждое поле представляет собой набор битов. Поле MAC адреса будем использовать для ведения истории. Для этого

разобьем поле на секции равной длины, в каждую секцию будем записывать двоичное представление номера коммутатора. Введем следующие обозначения:

- r – количество разрядов, выделенных под номер коммутатора;
- l – максимальное количество коммутаторов в истории (длина истории).
- k – количество разрядов в поле, отведенном под хранение истории.

Заметим, что номер коммутатора должен быть уникальным среди прочих, тогда количество разрядов, выделенных под номер коммутатора $r = \log_2 N$, где N – количество коммутаторов в сети.

Рассмотрим подробнее правила и таблицы (рис. 2), которые устанавливаются на коммутаторах в процессе работы алгоритма.

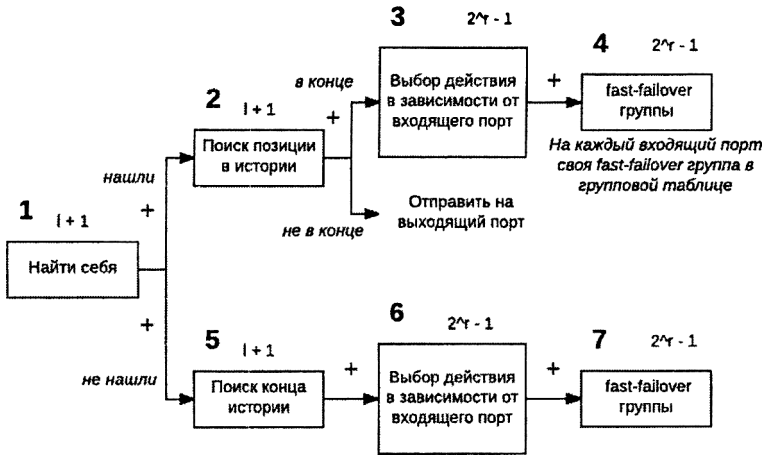

Рис. 2. Openflow таблицы и группы по обработке пакета

1. В первой таблице происходит определение нахождения текущего коммутатора в истории. Шаблон для каждого правила в данной таблице представляет последовательность длинною r для сравнения с полем MAC пакета. При этом каждая последовательность является двоичным представлением номера коммутатора и располагается со сдвигом $r*i$, где i – порядковый номер правила начиная с 0. То есть проверяются все возможные расположения номера коммутатора в истории. Все разряды, не входящие в текущую секцию

проверки, принимают подстановочные значения. Если коммутатор найден в истории, то перейти в таблицу 2, иначе в таблицу 7. Количество правил в данной таблице $l + 1$.

2. Определить положение коммутатора в истории. Если номер коммутатора встречается в середине истории – образовалась петля, и необходимо выполнить возврат по истории, отправив пакет на тот порт, через который он был получен. Для определения положения коммутатора в истории вводятся правила, у которых шаблон делится на три части: секция для сравнения с номером, разряды до секции, и разряды после секции. Разряды до секции имеют подстановочные значения, разряды после секции – нулевые. Таким образом, если будет найдено совпадение под один из таких шаблонов, то коммутатор в конце истории, иначе – образовалась петля. Для возврата по истории отправить на выходящий порт. Если номер коммутатора находит в конце истории, то передать таблице 3. Количество правил в данной таблице $l + 1$.

3. Необходимо определить следующую таблицу в зависимости от порта, на который пришел пакет. От этого зависит, какие пути ещё необходимо проверить, так как необходимо просмотреть лишь те соседние коммутаторы, номер которых больше, чем номер коммутатора, от которого только что пришел пакет. Такое поведение обусловлено тем, что перенаправление пакета происходит на минимальный номер, и если пакет вернулся обратно от какого-то коммутатора, то в коммутаторах с меньшим номером, чем у того от которого он вернулся, он уже побывал. Количество правил в данной таблице 2^r-1.

4. В зависимости от того на какой порт пришел пакет, он направляется в одну из fast-failover записей групповой таблицы, где пакет направляется на первый работающий порт из списка, упорядоченного по возрастанию номеров подключённых с их помощью коммутаторов, начиная с большего номера, чем номер коммутатора от которого пришел пакет. На данном этапе вводится 2^r-1 записей в групповой таблице.

5. Определение конца истории для записи собственного номера. Для этого шаблон у правил в рассматриваемой таблице проверяет позицию последней ненулевой секции. Проверка происходит секциями по r бит, предшествующие разряды имеют значение (*), а последующие - нулевое. Количество правил в данной таблице $l + 1$.

6. Аналогично 3.

7. В зависимости от того на какой порт пришел пакет, он направляется в одну из fast-failover записей групповой таблицы, где пакет направляется на первый работающий порт из списка, упорядоченного по возрастанию номеров подключённых с их помощью

коммутаторов, за исключением входного порта. На данном этапе вводится 2^r-1 записей в групповой таблице.

То есть в процессе работы алгоритма могут возникать две ситуации, fast-failover рассылка по определённым правилам, и возврат по истории в случае образования петли или исчерпания всех вариантов fast-failover рассылки в текущей вершине.

Приведем оценку количества правил при работе алгоритма предложенного в рамках работы.

Количсство правил M, которое требуется установить на один коммутатор:

$$M = 3l + 3 + 4*(2^r\text{-}1)$$

С учетом того, что $r = \log_2 N$, а $l = \frac{k}{r}$, где $k \leq N$, то

$$o(M) = o\left((2^{\log_2 N}\text{-}1)\right) = o(N)$$

Таким образом, для обеспечения соединения с одним коммутатором необходимо установить на все остальные коммутаторы сети такое количество правил, которое линейно зависит от их количества. Эту операцию нужно повторить для каждого коммутатора. В результате предложенный метод требует установки $o(N)$ правил, в то время как статическое резервирование – o(exp(N)).

Заключение

В рамках данной работы был разработан и исследован метод обеспечения защиты интегрированного управления в ПКС. В случае, когда необходимо достигнуть такого обеспечения отказоустойчивости, при котором сеть продолжает функционировать при наличии хотя бы одного рабочего маршрута, разработанный метод использует гораздо меньше правил, чем наивная реализация защиты, путем статического резервирования маршрутов.

Литература

1. Open Networking Foundation, « Software-Defined Networking (SDN) Definition» 26 Март 2015. [В Интернете]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition [Дата обращения: 18 Май 2016].

2. Open Networking Foundation, «OpenFlow Switch Specification» 26 Март 2015. [В Интернете]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.5.pdf. [Дата обращения: 14 Май 2016].

3. P. Goransson, Software Defined Networks A Comprehensive Approach, Massachusetts: Elsevier Inc., 2014.

4. Openvswitch.org, «Design Decisions In Open vSwitch,» 30 Март 2016. [В Интернете]. Available: http://openvswitch.org/support/dist-docs/DESIGN.md.txt. [Дата обращения: 18 Апрель 2016].

5. Sharma S. et al. «Fast failure recovery for in-band OpenFlow» networks //Design of reliable communication networks (drcn), 2013 9th international conference on the. 2013 IEEE, p 52-59.

6. OpenFlow, «Network Planning» 2011. [В Интернете]. Available: http://archive.openflow.org/wp/deploy-production-planning/. [Дата обращения: 2016].

7. S. Sharma, «In-band control, queuing, and failure recovery functionalities for openflow» IEEE Network , т. 30, № 1, pp. 106 - 112, 2016.

Янбулатов Р.А., Смелянский Р.Л.

Об одном подходе к эволюции традиционной сети в программно-конфигурируемую сеть[1]

Введение

Конец первой декады нового века ознаменовался осознанием кризиса компьютерных сетей [16]. Ответом на этот вызов стало появление принципиально нового подхода к их построению — программно-конфигурируемых сетей (ПКС) [17]. Технология ПКС предоставляет новые возможности по управлению сетями, повышению уровня автоматизации их администрирования, упрощает и удешевляет сетевое оборудование, снижает зависимость от производителя сетевого оборудования, открывает новые возможности для виртуализации сетей. Однако практика показала, что какими бы перспективными ни были новые технологии, если они требуют кардинальной смены инфраструктуры, то их внедрение обречено на неудачу [10].

Как правило, по техническим и экономическим причинам (например, высокая стоимость полного развёртывания сети), миграция сети в новую архитектуру не бывает одномоментной. На практике это всегда поэтапный процесс эволюции из старой архитектуры в новую.

В статье рассматривается задача построения методики перехода от сети с традиционной архитектурой TCP/IP к сети с ПКС архитектурой. В работе предложен способ выбора маршрутизаторов в традиционной сети для замены их на ПКС коммутаторы так, чтобы за минимальное число этапов замены взять под ПКС управление наибольшее число потоков данных в сети.

1. Обозначения

Здесь и далее будем представлять сеть в виде графа $G = (V, E)$, вершины которого суть маршрутизаторы, соединённые некоторым образом между собой. Где $V = \{v_1, ..., v_n\}$ — множество вершин этого графа, $E \subset V \times V$ — множество неориентированных рёбер $e = \{v_i, v_j\}$, соответствующих паре непосредственно соседних маршрутизаторов, то есть между которыми есть канал.

[1] Работа выполнена при финансовой поддержке РФФИ, грант № 14-07-00625/16.

Воспользуемся определением потока из [9], где поток определён как двунаправленная последовательность пакетов со следующими параметрами:

Адрес источника;

Адрес назначения;

Порт источника для UDP и TCP;

Порт назначения для UDP и TCP;

Тип и код сообщения для ICMP;

Номер протокола IP;

Сетевой интерфейс (параметр ifindex SNMP);

IP Type of Service.

Критерии оптимизации выбора OSPF маршрутизаторов для замены могут быть разные. Например, это может быть количество потоков, либо объём данных по всем потокам, проходящих через OSPF маршрутизатор. Чтобы обобщить задачу на оба случая, введём понятия веса для потока, вершины и маршрута.

Определение 1. Вес потока будем полагать либо равным сумме длин пакетов его составляющих в байтах, если требуется взять под ПКС управление максимальное число наиболее «тяжелых» потоков, либо равным 1, если требуется максимизировать лишь число потоков.

Определение 2. Вес вершины графа G равен суммарному весу потоков, прошедших через соответствующий маршрутизатор за определённый период времени.

Определение 3. Весом маршрута *Path* назовём суммарный вес потоков, маршрут которых в точности совпадает с *Path* (если таких потоков нет, то маршрут имеет нулевой вес).

2. Постановка задачи

Дана локальная сеть с архитектурой TCP/IP, канальный уровень которой представлен технологией Ethernet. В сети есть

абонентские машины, на которых выполняются приложения;

маршрутизаторы, работающие по протоколу OSPF (или IS-IS). Будем называть их OSPF маршрутизаторы.

Также задано число ПКС коммутаторов, имеющихся в распоряжении для замены TCP/IP маршрутизаторов.

Предполагаем, что ПКС коммутаторы могут работать как в режиме ПКС коммутатора, так и OSPF маршрутизатора. Требуется определить, какие OSPF маршрутизаторы следует заменить на имеющиеся ПКС коммутаторы так, чтобы взять под ПКС управление наибольшее число потоков в сети.

В работе эта задача решается в трёх разных постановках, различающихся по детальности собранной информации о потоках:

В первой постановке считаются известными маршрут каждого потока, прошедшего в сети за определенный представительный период времени и вес потоков.

Во второй постановке для каждой вершины, соответствующей OSPF маршрутизатору, известен её вес.

В третьей постановке информация о потоках не используется.

3. Решение задачи в первой постановке

3.1. Необходимость информации о маршрутах потока

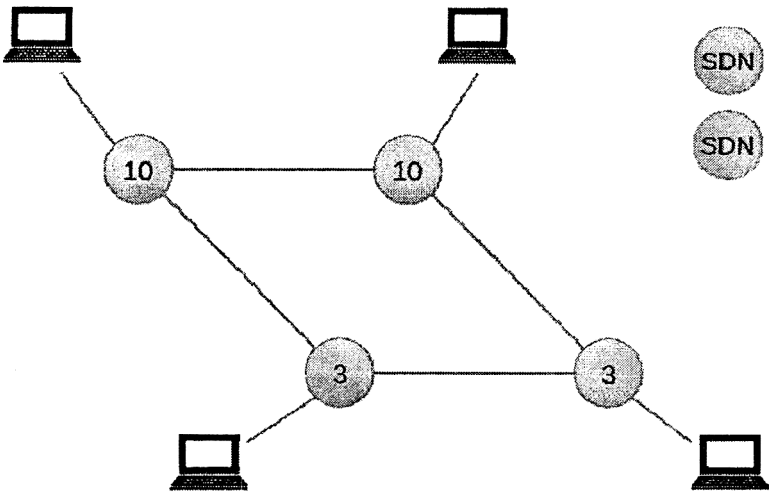

Рис. 1: Маршруты потоков неизвестны

Допустим, нам известны маршруты потоков и вес каждого маршрута. Почему полезно учитывать маршрут каждого потока? Рассмотрим сеть, изображённую на рис. 1, в которой 4 маршрутизатора, 2 ПКС коммутатора для замены и известно, сколько потоков прошло через каждый маршрутизатор. Без учёта маршрута каждого потока, логичным выглядит заменить маршрутизаторы, через которые прошло по 10 потоков. Однако, зная маршруты потоков (рис. 2), становится очевидно, что достаточно заменить маршрутизаторы A и C, чтобы взять под ПКС управление все потоки в этой сети (13), тогда как при предыдущих рассуждениях было покрыто всего 10 потоков.

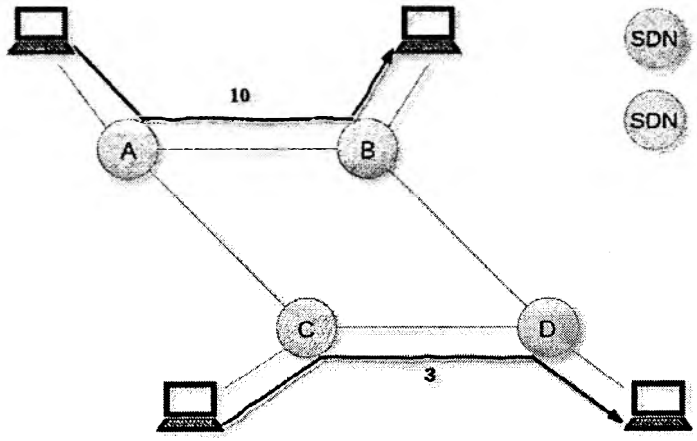

Рис. 2: Маршруты потоков известны

Разберём на этом же примере удобное представление распределения потоков в виде матрицы.

3.2. Алгоритм полного перебора

Рассмотрим некоторый поток, пусть $Path_i = (v_{i_1}, ..., v_{i_n})$ — его маршрут, и $D = \{Path_1, ..., Path_l\}$ — совокупность всех маршрутов потоков в этой сети. Через $\mathrm{N}_0^{s,n}$ обозначим множество матриц с s строками, n столбцами и элемснтами из $\mathrm{N}_0 = \{0,1,2,3,...\}$. Сопоставим паре (V, D) матрицу $M \in \mathrm{N}_0^{|V|,|D|}$, для которой значение M_{ij} равно весу маршрута, если $v_i \in Path_j$ и 0 иначе. Будем говорить, что i-я строка матрицы M покрываст сё j-й столбец, если $M_{ij} > 0$, то есть $v_i \in Path_j$.

Таким образом, для приведённого примера получается следующая таблица:

	$Path_1$	$Path_2$
A	10	0
B	10	0
C	0	3
D	0	3

И строки, соответствующие вершинам A и C, покрывают все столбцы данной таблицы.

Чтобы посчитать, сколько потоков проходит через некоторое подмножество вершин $X = \{v_{i_1}, ..., v_{i_n}\}, X \subseteq V$, можно использовать следующий алгоритм из n шагов, где n — мощность множества X:

Шаг 1. В матрице M выбирается строка i_1, соответствующая вершине v_{i_1} из X. Сумма всех значений в этой строке добавляется к итоговому результату. Из матрицы M вычёркивается строка i_1 и все столбцы, на пересечении которых со строкой i_1 стоят не 0; получается матрица M_1.

. . .

Шаг s. В матрице M_{s-1} выбирается строка i_s, соответствующая вершине v_{i_s} из X. Сумма всех значений в этой строке добавляется к итоговому результату. Из матрицы M_s вычёркивается строка i_s и все столбцы, на пересечении которых со строкой i_s стоят не 0; получается матрица M_s.

Будем считать, что известен маршрут каждого потока и вес каждого маршрута. Тогда, при решении полным перебором, мощность множества вариантов выбора будет равна

$$C_n^k = \frac{n!}{k!(n-k)!},$$

где k – число маршрутизаторов для замены, n – число маршрутизаторов в сети. Для каждого варианта выбора потребуется по таблице пересчитать, сколько потоков захватывается кандидатами для замены и найти максимальный вариант. Понятно, что переборный алгоритм в общем случае работает не за полиномиальное время.

Покажем, что данная задача в принципе не решается за полиномиальное время, если $P \neq NP$.

3.3. Обоснование NP-трудности задачи

Через $B^{s,n}$ обозначим множество матриц с s строками, n столбцами и элементами из $B = \{0,1\}$. Для обоснования свойств нашей задачи сначала сформулируем ряд вспомогательных вопросов.

Вопрос 1. Какое максимальное число потоков можно контролировать с помощью k ПКС коммутаторов?

Вопрос 2. Хватит ли k ПКС коммутаторов, чтобы контролировать все заданные потоки в заданной сети?

Вопрос 3. Хватит ли k строк в матрице $M \in B^{p,s}$, чтобы покрыть все столбцы этой матрицы?

Вопрос 4. Хватит ли k столбцов в матрице $M \in B^{p,s}$, чтобы покрыть все строки этой матрицы?

Вопрос 5. Для заданного конечного множества $\mathcal{N} = \{\alpha_1, ..., \alpha_s\}$, семейства его подмножеств $\mathcal{S} = \{\mathcal{N}_1, ..., \mathcal{N}_p\}$ и целого числа k ответить на вопрос, существует ли семейство $\mathcal{C} \subseteq \mathcal{S}$ мощности k такое, что $\bigcup_{\mathcal{N}_i \in \mathcal{C}} \mathcal{N}_i = \mathcal{N}$.

Заметим, что вопрос 5 является проблемой разрешимости [11] для известной задачи о покрытии множества, для которой доказано, что она является NP-трудной, а соответствующая задача о разрешении является NP-полной [3]. Далее с помощью последовательной цепочки сводимостей от вопроса 5 к вопросу 1 легко показать [18], что задача 5 полиномиально сводится к нашей задаче.

3.4. Градиентный алгоритм

В связи с тем, что нахождение точного решения является трудоёмкой задачей, воспользуемся жадным алгоритмом, на каждом шаге которого выбирается вершина с максимальным весом. Опишем этот алгоритм выбора s маршрутизаторов для замены на ПКС коммутаторы.

Алгоритм:

Для каждой вершины пересчитываем её вес.

Заменяем маршрутизатор, соответствующий вершине с максимальным весом. Из множества рассматриваемых потоков убираем те потоки, которые проходят через эту вершину.

$s := s - 1$. Если s = 0, алгоритм закончен.

Иначе перейти к шагу 1.

Для корректной работы данного алгоритма удалим из рассмотрения те вершины, через которые не проходит ни один поток. Сопоставим паре (V, D) матрицу $M \in \mathrm{B}^{|V|,|D|}$, для которой $M_{ij} = 1$ тогда и только тогда, когда $v_i \in Path_j$. В матричной постановке на каждом шаге градиентного алгоритма в матрице выбирается и включается в покрытие такая строка, которая покрывает наибольшее число ещё непокрытых столбцов. Следующая теорема даёт верхнюю оценку количества шагов в градиентном алгоритме, необходимых для покрытия всех столбцов с заданной «густотой».

Теорема 1 ([14]). Пусть для вещественного γ, $0<\gamma\leq 1$, в каждом столбце матрицы M, $M \in \mathrm{B}^{p,s}$ имеется не меньше, чем γp единиц. Тогда покрытие матрицы M, получаемое с помощью градиентного алгоритма, имеет длину не больше, чем $\left\lceil \frac{1}{\gamma} \ln^{+}(\gamma s) \right\rceil + \frac{1}{\gamma}$.

$$ln^{+}x = \begin{cases} lnx, & x \geq 1 \\ 0, & 0 < x < 1 \end{cases}$$

Применительно к нашей задаче $M \in \mathrm{B}^{|V|,|D|}$ и данная теорема даёт верхнюю оценку количества маршрутизаторов, которые потребуется заменить на ПКС коммутаторы, чтобы взять под контроль все потоки, количество hop'ов которых не менее $\gamma \cdot |V|$.

4. Решение задачи во второй постановке

4.1. Алгоритм разрезания

Идея данного подхода, впервые предложенного в [4], заключается в том, что ПКС коммутаторы расставляются в сети таким образом, чтобы разбить OSPF домен на 2 или более поддомена. Тогда все потоки, которые проходят внутри поддоменов остаются нетронутыми, в то время как потоки между поддоменами обязательно проходят хотя бы через один ПКС коммутатор.

Задача разделения графа его вершинами в теории графов известна как Vertex separator problem. Основываясь на модели vertex separator из работы [2], нами была сформулирована модель разделения сети в терминах задачи булева линейного программирования (БЛП). При построении модели использовалась лишь частичная информация о

характерс трафика в предложенной сети — вес каждой вершины (который, напомним, может характеризовать как количество потоков, прошедших через данную вершину, так и размер пакетов их составляющих). Однако заметим, что при необходимости можно построить модель, которая не учитывает даже этих данных, а использует только топологию сети. В такой модели целью работы алгоритма будет разбиение графа на подграфы сопоставимых размеров.

4.2. Постановка в терминах теории графов

Пусть задан связный неориентированный граф $G = (V, E)$:

$V = \{v_1, ..., v_n\}$ — множество вершин этого графа; |V | = n

$E \subset V \times V$ — множество неориентированных рёбер $\{v_i, v_j\}$

c_j — вес j-й вершины

$s \leq n$ — ограничение на число заменяемых вершин

Задача

Найти разбиение множества V на 3 непересекающихся множества $\{A, S, B\}$ таких, что:

Для $\forall v_i \in A$ и $\forall v_j \in B$ ребро $\{v_i, v_j\}$ не содержится в E.

$|S| \leq s$ — ограничение на длину сечения;

$\sum_{v_j \in S} c_j$ максимальна.

Заметим, что условие 1 эквивалентно тому, что между любыми двумя вершинами v_i, v_j, такими что $v_i \in A$ и $v_j \in B$, не существует маршрута, который бы не проходил через сечение S.

Условие 3 определяется тем, что мы хотим максимизировать вес потоков, проходящих через ПКС коммутаторы.

В [18] можно найти соответствующую задачу булева линсйного программирования, которая решает данную задачу.

5. Решение задачи в третьей постановке

5.1. Алгоритм удаления циклов

Идея данного алгоритма предложена в [7] и основана на гипотезе о том, что вершинам, через которые проходит много циклов, соответствуют маршрутизаторы, через которые проходит много трафика.

Определение 4. Базис циклов — минимальный набор циклов, который позволяет любой эйлеров подграф представить, как симметрическую разность базисных циклов.

Определение 5. Цикломатическое число графа — минимальное число рёбер, которые надо удалить, чтобы граф стал ациклическим. Существует соотношение: $p_1(G) = |E| - |V| + p_0(G)$, где $p_1(G)$ – цикломатическое число, $p_0(G)$ – число компонент связности, $|E|$ – число рёбер и $|V|$ – число вершин.

Поскольку каждый граф $G = (V, E)$ имеет хотя бы один базис циклов, любой цикл в G может быть представлен как линейная комбинация фундаментальных циклов, составляющих базис.

Стратегия данного алгоритма основывается на утверждении, что нахождение множества фундаментальных циклов – циклов, составляющих базис циклического пространство графа, – может быть осуществлено эффективно.

Для того чтобы построить базис циклов, выделяется некоторое остовное дерево. При добавлении любого ребра, не входящего в это остовное дерево, образуется цикл. Семейство таких циклов образует базис циклического пространства этого графа. Построив базис циклов, «жадно» заменяются те вершины, через которые проходит наибольшее количество базисных циклов.

Алгоритм ([7]):

Построить базис циклов B.

Заменить вершину v, через которую проходит наибольшее количество циклов.

Удалить из рассмотрения эту вершину (заменить ПКС коммутатором).

Удалить из B все циклы, которые проходят через v.

Если больше нет ПКС коммутаторов, закончить.

Если в B остались циклы, вернуться к шагу 2.

Если в графе остались циклы, вернуться к шагу 1.

5.2. Сведение задачи удаления циклов к задаче целочисленного программирования

В алгоритме, предложенном выше присутствует неоднозначность при построении базиса циклов: остовные деревья можно строить по-разному (число остовных деревьев в полном графе на n вершинах равно n^{n-2}). А следовательно, неоднозначен и выбор замсняемых вершин.

Задача о том, какое минимальное количество вершин понадобится заменить, чтобы удалить все циклы в графе, известна в теории графов как Feedback vertex set problem и является NP-полной задачей [1]. Для нахождения оптимизационного решения данной задачи, построим соответствующую задачу целочисленного программирования.

Для каждой вершины $v \in G$ введём переменную b_v, которая равна 1, если эту вершину следует заменить и 0 иначе. Для того, чтобы граф получился ациклическим, эти переменные должны удовлетворять следующему ограничению:

$$\text{Для } \forall \text{ цикла } C \subseteq G \quad \sum_{v \in C} b_v \geq 1$$

То есть в любом цикле в G хотя бы одну вершину следует заменить для того, чтобы этот цикл «распался».

При этом минимизируется число заменяемых вершин. Из этих рассуждений вытекает следующая задача булева линейного программирования:

$$\text{Минимизировать: } \sum_{v \in G} b_v$$

Так, чтобы:

$$\text{Для } \forall \text{ цикла } C \subseteq G, \sum_{v \in C} b_v \geq 1$$

Заключение

В рамках данной работы были исследованы различные методы эволюции традиционной сети в ПКС сеть. Были разработаны и исследованы алгоритмы эволюции, различающиеся стратегией и объёмом входных данных (а именно, статистикой, собранной в сети до запуска этих алгоритмов).

В ходе работы были достигнуты следующие результаты.

Для стратегий разрезания графа и удаления циклов:

Были сформулированы (и программно реализованы) соответствующие задачи целочисленного линейного программирования.

Для задачи, когда задан полный маршрут каждого потока:

Обоснована NP-трудность поставленной задачи.

Предложен соответствующий жадный алгоритм.

Получена верхняя оценка необходимого количества ПКС коммутаторов для контроля всех предоставленных потоков.

Литература

1. Approximating minimum feedback sets and multicuts in directed graphs / Guy Even, J Seffi Naor, Baruch Schieber, Madhu Sudan // Algorithmica. — 1998. — Vol. 20, no. 2. — P. 151–174.

2. Balas Egon, de Souza Cid C. The vertex separator problem: a polyhedral investigation // Mathematical Programming. — 2005. — Vol. 103, no. 3. — P. 583–608.

3. Combinatorial optimization / Bernhard Korte, Jens Vygen, B Korte, J Vygen. — Springer, 2002.

4. Divide and conquer: Partitioning OSPF networks with SDN / Marcel Caria, Tamal Das, Admela Jukan, Marco Hoffmann // Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on / IEEE. — 2015. — P. 467–474.

5. Incremental SDN deployment in enterprise networks / Dan Levin, Marco Canini, Stefan Schmid, Anja Feldmann // ACM SIGCOMM Computer Communication Review. — 2013. — Vol. 43, no. 4. — P. 473–474.

6. Insights on SDN migration trajectory / Tamal Das, Marcel Caria, Admela Jukan, Marco Hoffmann // Communications (ICC), 2015 IEEE International Conference on / IEEE. — 2015. — P. 5348–5353.

7. Markovitch Michael, Schmid Stefan. Shear: A highly available and flexible network architecture: Marrying distributed and logically centralized control planes // Proc. 23rd IEEE ICNP. — 2015.

8. Panopticon: Reaping the benefits of partial sdn deployment in enterprise networks / Dan Levin, Marco Canini, Stefan Schmid, Anja Feldmann. — Die Professoren der Fakultät IV, Elektrotechnik und Informatik, 2013.

9. Wikipedia. Netflow // Википедия, свободная энциклопедия. — 2012. — URL: https://ru.wikipedia.org/wiki/Netflow.

10. Wikipedia. ATM // Википедия, свободная энциклопедия. — 2014. — URL: https: //ru.wikipedia.org/wiki/ATM.

11. Wikipedia. Проблема разрешимости // Википедия, свободная энциклопедия. — 2015. — URL: https://ru.wikipedia.org/wiki/Проблема_разрешимости.

12. Гэри Майкл, Джонсон Дэвид. Вычислительные машины и труднорешаемые задачи. — Мир, 1982. — Т. 3.

13. Дискретная математика и математические вопросы кибернетики / Под ред. С.В. Яблонского и О.Б. Лупанова. — М.: Наука, 1974. — Т. 1.

14. Ложкин С.А. Основы кибернетики. — 2003.

15. Новикова Н.М. Основы оптимизации // Курс лекций в МГУ. — 1999.

16. Смелянский Р.Л. Проблемы современных компьютерных сетей // Труды XIX Всероссийской научно-методической конференции Телематика. — 2012. — URL: http://tm.ifmo.ru/tm2012/src/024e.pdf

17. Смелянский Р.Л. Программно-конфигурируемые сети // Открытые системы. — 2013. — URL: http://www.osp.ru/os/2012/09/13032491/.

18. Янбулатов Р.А. Исследование различных способов эволюции традиционной сети в программно-конфигурируемую сеть — курсовая работа //Московский государственный университет имени М.В.Ломоносова, 2016 [PDF] (https://cloud.lvk.cs.msu.su/index.php/s/xV8isRmPNtHcFP1).

Колосов А.М., Смелянский Р.Л.

Исследование подходов к построению информационно-ориентированных сетей[1]

1. Введение

В настоящее время компьютерные сети архитектуру ориентированная на адресацию устройств, хостов, в таких сетях данные хранятся на узлах, обращение к которым происходит по их адресам (далее такие сети называются традиционными сетями). Однако пользователю не всегда может быть известен адрес узла, а если и известен, то узел может быть недоступен. Кроме того, если пользователь хочет найти какие-либо данные, точное месторасположение которых он не знает, - он обращается к поисковым системам и затем вручную отбирает результаты поиска. Но поисковые системы индексируют не все ресурсы, предоставляющие доступ к данным или, индексируют их не полностью. Поэтому в итоге искомые данные могут так и не быть найдены. Таким образом, в традиционных сетях получение доступа к информации предполагает знание адреса узла, на котором данная информация хранится, однако адрес и информация никак между собой не связаны, поэтому и приходится применять некоторые вспомогательные средства - такие как поисковики. То есть, говоря о разделенном доступе к ресурсам в традиционных сетях, имеется в виду разделенный доступ к физическим ресурсам, однако сейчас акцент смещается и теперь это разделенный доступ к информационным ресурсам.

Решением описанной выше проблемы традиционных сетей является переход к концепции сетей, ориентированных на информацию, в которых адресуются не устройства, а информация. Такие сети будем называть информационно-ориентированными сетями (Information-Centric Networking, ICN [1]). Если в ICN сетях есть хотя бы по одному экземпляру искомых данных, то пользователи их получат, ничего не зная об их расположении. Помимо этого, в ICN сетях присутствует механизм кэширования на узлах, который позволяет значительно снизить нагрузку на сеть и обеспечивать эффективный групповой работы. Также в ICN сетях на уровне самой сети обеспечивается безопасность данных при помощи подписей их владельцев.

Поскольку такая концепция ICN сети позволяют эффективно работать с данными, возникает вопрос как строить такие сети, какова должна быть их архитектура, нужно ли разрабатывать ее с нуля или

[1] Работа выполнена при финансовой поддержке РФФИ, грант № 14-07-00625/16.

можно использовать существующие технологии - такие как технологии программно-конфигурируемых сетей (Software-Defined Networking, SDN [2])?

2. SDN – Программно-конфигурируемое сети

SDN сети – это сети, в которых управление передачей данных и собственно передача данных разделены. В SDN сетях есть два типа устройств: первый тип устройств, SDN-коммутатор, осуществляет исключительно передачу данных по правилам, устанавливаемым вторым типом устройств, SDN-контроллером, которому известен состав и состояние сети и который по сути может управлять ею путем установки правил на SDN-коммутаторы. Таким образом, в SDN сетях появляется некоторая логическая централизация, реализуемая в виде SDN-контроллера, который может отличать запросы к данным, доступным в ICN сети, от всех остальных запросов и отправлять эти запросы выделенному узлу, где в специальной БД (далее эта БД называется ICN-БД) может скапливаться и откуда может извлекаться информация обо всех данных, доступных в ICN сети (далее этот узел называется ICN-Интерфейсом Базы Данных, ICN-ИБД). Поскольку из соображений безопасности ICN-ИБД имеет смысл делать доступным только через SDN-контроллер - в дальнейшем предполагается, что оба эти элемента работают в рамках одного узла сети – то есть между ними нет хопа.

Одной из проблем ICN сетей является проблема поиска узлов, содержащих запрошенные данные (далее такие узлы, как с кэшированием, так и без него, называются ICN-узлами). Одним из решений данной проблемы является использование технологий SDN сетей, в которых архитектурно присутствует центральный узел, SDN-контроллер. SDN-контроллер, взаимодействуя с ICN-ИБД, который индексирует данные и знает, где они расположены, может настраивать SDN-коммутаторы на пути от пользователя до источника запрошенных им данных - то есть ICN-узла, на котором эти данные хранятся не в кэше. Также, если дополнительно хранить в ICN-БД информацию о кэшах на ICN-узлах, то можно строить наиболее короткие маршруты. Более того, когда SDN-контроллер используется совместно с ICN-ИБД, появляется возможность при обнаружении новых данных заранее информировать ICN-узлы об их источниках и таким образом полностью решить задачу поиска источников, запрошенных данных. Однако тогда возникает другая проблема, связанная с переполнением таблиц с информацией об источниках запрошенных данных на ICN-узлах, поэтому далее в экспериментальном исследовании предполагается, что кэширование на ICN-узлах отсутствует.

3. Концепция Information Centric Network

ICN – это концепция, идея которой в том, что в отличие от традиционных сетей, где центральным элементом являются узлы, на которых расположены данные, в ICN центральным элементом являются сами данные. То есть в отличие от традиционных сетей, где адресация производится по адресам, в ICN адресация производится по именам данных. То есть пользователь может обратиться к такой сети за данными и получить их, при этом ничего не зная о том, где они расположены.

ICN сети могут быть использованы для организации Сетей доставки контента (Content Delivery Network, CDN), в которых копии контента распределяются по выделенным узлам сети и затем при реализации запроса к этому контенту возвращается та его копия, которая находится на узле с наименьшим числом хопов до отправившего запрос пользователя. Также CDN позволяет снизить нагрузку на ресурсы, предоставляющие доступ к данным, например, веб-сайты, логически организуя вокруг них кольцо, которое кэширует данные, к которым чаще всего обращаются пользователи, и поэтому может отвечать им на запросы, таким образом защищая ресурс от перегрузок.

Также ICN сети могут быть использованы для организации систем, в которых требуется эффективная многоцелевая доставка контента – то есть доставка контента множеству пользователей без многократного отправления его копий по совпадающим участкам маршрутов от поставщика к потребителям; например, систем стриминга (Twitch) или доступа к видео по запросу (YouTube). В ICN сетях данные и узлы независимы, поэтому при условии, что данные кэшируются, после обращения первого потребителя к данным, они будут расположены на всех ICN-узлах, составляющих кратчайший маршрут от первого потребителя до поставщика, и для всех последующих потребителей, у которых кратчайший маршрут до поставщика пересекается с тем же маршрутом у первого потребителя, данные будут возвращаться с ближайшего из крайних ICN-узлов, составляющих этот маршрут.

4. Математическая модель ICN сети

Разделим произвольную ICN сеть, в которой на одном случайно выбранном ICN-узле находится Потребитель, а на другом ICN-узле, также случайно выбранном, находится Поставщик, на следующие сегменты (рисунок 1):

1) множество узлов, мощностью N, связанных с остальной сетью только через ICN-узел, на котором находится Потребитель

2) множество узлов, мощностью K, связанных с остальной сетью только через ICN-узел, на котором находится Поставщик

3) множество остальных узлов, мощностью V

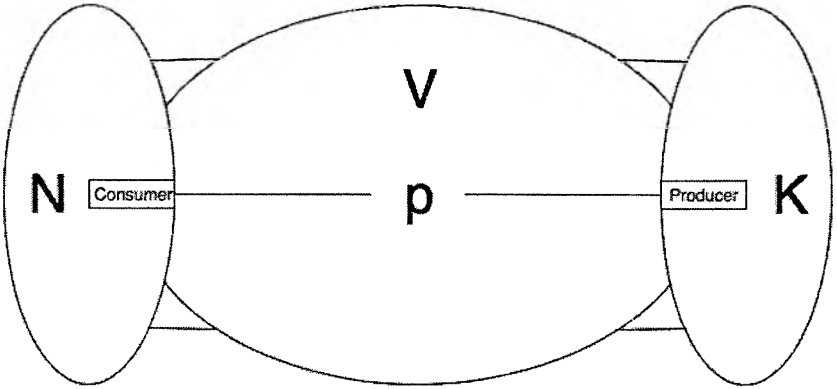

Рис. 1. Топология рассматриваемой сети

1) множество узлов, связанных с остальной сетью только через ICN-узел, на котором находится Потребитель, мощностью N.

2) множество узлов, связанных с остальной сетью только через ICN-узел, на котором находится Поставщик, мощностью K.

3) множество остальных узлов, мощностью V.

Для определенности будем считать, что Потребитель и Поставщик относятся к V и не относятся соответственно к N и K. Заметим, что для многих случаев множества мощностью N и K будут пустыми. Для того, чтобы определить эти множества, можно воспользоваться следующим алгоритмом - не ограничивая общности, рассмотрим его на примере определения множества мощностью N: убираем из графа вершину, на которой находится Потребитель и все ребра, инцидентные ей (далее полученный граф называется оставшимся). Далее перебираем все вершины в оставшемся графе. Для каждой из них мы проверяем принадлежит ли она множеству мощностью N, пытаясь построить для нее путь до Поставщика (например, используя алгоритм Дейкстры) - если и только если такой путь в оставшемся графе есть, то он будет найден и это будет означать, что вершина не принадлежит множеству мощностью N.

Обозначим кратчайшее расстояние между ICN-узлами Потребителя и Поставщика как p. Заметим, что p не превосходит диаметра сегмента V (обозначим его как d) в случае если сегмент V связный, в противном случае будем считать, что значение p не определено. Диаметр - наибольшее расстояние между любыми парами вершин. Расстояние между вершинами - длина кратчайшего пути между ними. Чтобы найти диаметр графа, сначала находят кратчайшие пути между всеми парами вершин. Наибольшая длина среди найденных путей и есть диаметр графа.

В качестве критерия эффективности (меры) выбрано отношение числа ICN-узлов, задействованных для реализации запроса, к общему числу ICN-узлов в ICN сети (далее это соотношение называется коэффициентом утилизации ресурсов ICN сети).

Выбор этой меры обусловлен тем, что поскольку время реализации запроса выражается в виде произведения числа ICN-узлов, задействованных для реализации запроса, на время, которое каждый из этих узлов тратит в среднем на обработку запроса, то для решения задачи уменьшения времени реализации запроса, в первую очередь следует уменьшить количество ICN-узлов, задействованных для реализации запроса, и только затем уменьшать время обработки запроса на каждом из задействованных ICN-узлов. Общее число ICN-узлов сети в знаменателе меры позволяет ей быть независимой от масштаба ICN сети, поскольку делает меру безразмерной.

Поскольку в ICN сети, построенной с помощью технологий традиционных сетей, нет ни одного узла, которой обладал бы информацией обо всей сети, то такая сеть является распределенной сетью, поэтому для поиска источника данных, требуется задействовать все ICN-узлы, кроме тех, которые находятся за источником данных.

Таким образом, в случае построение ICN сетей с помощью технологий традиционных сетей, коэффициент утилизации равен:

$$k_{trad} = \frac{V + N}{N + V + K} \qquad (1)$$

Потребитель отправляет Интерес в сегменты N и V, в которых он распространяется по всем ICN-узлам данных сегментов. При достижении Поставщика, распространение прекращается, поэтому в сегменте K Интерес распространяться не будет.

Поскольку в ICN сети, построенной с помощью технологий SDN сетей, присутствует узел, который обладает информацией обо всей сети, SDN-контроллер, то такая сеть является централизованной сетью. Поэтому в случае наличия выделенного узла, ICN-ИБД, где в ICN-БД может скапливаться и откуда может извлекаться информация обо всех данных, доступных в ICN сети, SDN-контроллер, взаимодействуя с этим ICN-ИБД, может настраивать SDN-коммутаторы таким образом, что запрос пойдет по кратчайшему пути между Потребителем и Поставщиком.

Таким образом, в случае построение ICN сетей с помощью технологий SDN сетей, коэффициент утилизации равен:

$$k_{sdn} = \frac{p}{N + V + K} \qquad (2)$$

Потребитель отправляет запрос, который поступает на SDN-контроллер, который в результате взаимодействия с ICN-ИБД,

настраивает SDN-коммутаторы для передачи запроса по кратчайшему пути, содержащему p ICN-узлов.

Рассмотрим отношение получившихся коэффициентов:

$$\frac{k_{sdn}}{k_{trad}} = \frac{\frac{p}{N+V+K}}{\frac{V+N}{N+V+K}} = \frac{p}{V+N} \quad (3)$$

$$p \le d \le V+N \quad (4)$$

$$\frac{p}{V+N} \le 1 \quad (5)$$

$$\frac{k_{sdn}}{k_{trad}} \le 1 \quad (6)$$

Таким образом, обосновано, что ICN сети, использующие технологий SDN сетей, гарантированно утилизирует сеть не сильнее чем ICN сети, использующие технологии традиционных сетей, независимо от топологии сети и выбора узлов расположения Потребителя и Поставщика.

Далее в работе проводится экспериментальное исследование, подкрепляющее гипотезу для конкретной топологии.

Отдельно стоит отметить случай, когда в (4) наблюдается равенство:

$$p = d = V+N \quad (7)$$

В этом случае множество узлов с мощностью N оказывается пустым, а множество узлов с мощностью V состоит в точности из p узлов - то есть получается, что эти два множества представляют из себя цепочку узлов от узла Потребителя до узла Поставщика, возможно продолжающуюся за Поставщиком, и есть только один вариант их связи друг с другом, что по сути является вырожденным случаем и для того, чтобы не ограничивать общность понятия «более эффективно» делается допущение, что этот случай исключается из рассмотрения в силу своей без альтернативности. Вырожденным случаем цепочки является цепочка из одного узла, на котором располагаются и Потребитель, и Поставщик. В этом случае p равно единице, поскольку расстояние между узлами рассчитывается как число узлов, составляющих кратчайший путь между ними. Мощность V также равна единице, поскольку множества мощностью N и K оказываются пустыми - в этом можно убедиться, проделав действия, описанные в алгоритме определения этих множеств. Отдельно стоит отметить, что в случае, когда Потребитель и Поставщик находятся на одном и том же узле и общее число узлов ICN сети больше одного, в соответствии с алгоритмом получается, что все другие вершины будут учтены дважды(и в множестве мощностью N, и в

множестве мощностью K) и таким образом, значение коэффициента утилизации ресурсов ICN сети в случае построение на основе технологий традиционных сетей (формула 1) не будет отражать суть происходящего в ней, однако в отношении коэффициентов (формула 3) в знаменателе будут учтены все узлы, причем каждый ровно один раз. В то же время, кажется логичным сначала провести поиск среди локальных ресурсов, а уже затем обращаться к сетевым ресурсам, таким образом уменьшая количество узлов, задействованных для реализации запроса до одного. Таким образом, результат измерения зависит от выбранного порядка работы с локальными ресурсами - в экспериментальном исследовании, которое представлено далее, считается, что обращение к сетевым ресурсам происходит независимо от расположения данных, поскольку речь все-таки не идет о локальном взаимодействии.

5. Программная реализация

5.1. Архитектура модели

Для построения имитационных моделей используется программный продукт для виртуализации операционных систем - гипервизор ESXi [13], на котором создаются виртуальные машины, на которых запускаются реализации ICN.

Для моделирования была выбрана следующая топология:

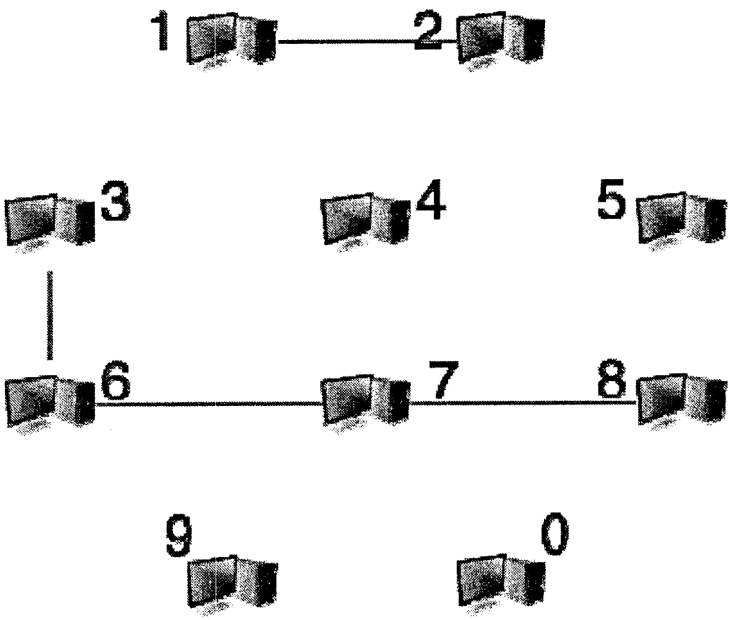

Рис. 2. Топология сети

5.2. Описание модели на основе традиционной сети

Для построения ICN на основе технологий традиционных сетей прежде всего нужно определить на каком уровне сетевого стека традиционной сети проводить построение ICN: на сетевом или на канальном.

В случае построения ICN на сетевом уровне, можно выделить следующие достоинство и недостаток.

Достоинство заключается в том, что появляется возможность взаимодействовать между несвязанными друг с другом сегментами ICN сети путем туннелирования через традиционную сеть.

Недостаток заключается в том, что поскольку сетевой уровень предполагает наличие иерархической системы имен, то происходит наложение одной иерархической системы на другую, что создает как логическую, так и фактическую избыточность, выражающуюся в увеличенном размере сообщения и соответственно увеличении времени его обработки на сетевых устройствах.

В случае построения ICN сети на канальном уровне, картина получается противоположная: то есть утрачивается возможность взаимодействия между несвязанными друг с другом сегментами ICN сети, но в то же время в сообщениях отсутствуют поля, связанные с сетевым уровнем.

Таким образом, выборе уровня организации зависит от связности сегментов ICN сети - в случае, если сегменты напрямую между собой не связаны, следует выбрать сетевой уровень, если иначе - канальный уровень.

Для моделирования на основе технологий традиционных сетей используется система туннелей на канальном уровне и таким образом, говоря о пространстве имен, можно провести аналогию между иерархической ICN системой имен поверх плоской системы имен, использующей MAC-адреса и такой же иерархической системой имен в IP поверх той же самой плоской.

Для подключения к ICN сети на основе традиционной сети нового ICN-узла, требуется создать туннель между этим ICN-узлом и любым другим ICN-узлом ICN сети - это можно сделать, например, с использованием VLAN.

Для моделирования используется виртуальные машины под управлением ОС Ubuntu версии 14.04.1 с предустановленными компонентами:

1) CCNx_Distillery [14]
2) ccnxHelloWorld [15]
3) ccnxwireshark [16]

5.3. Описание модели на основе SDN

Для построения ICN на основе SDN технологий требуется организовать взаимодействие между ICN и SDN. Это взаимодействие можно организовать разными способами [9]: их можно разделить на два класса - подразумевающие внесение изменений в протокол OpenFlow [17] и не подразумевающие. И поскольку изменение протокола крайне затруднительно, то в конечном итоге предлагается подход, когда протокол OpenFlow не изменяется, однако у него появляется так называемый второй слой для работы конкретных приложений (second application-specific layer) - в данном случае ICN-приложений. Этот уровень реализуется при помощи дополнительного канала для взаимодействия между коммутаторами и контроллером параллельно каналу взаимодействия через OpenFlow.

Таким образом, требуется реализовать средство, состоящее из расширения для ICN-узла, позволяющего ему перенаправлять Интересы и Данные коммутатору и соответственно приложения для контроллера, которое сможет обрабатывать эти сообщения. Такое средство существует и называется NDNFlow [4] - далее оно используется в исследовании.

Для подключения к ICN-сети на основе SDN технологий нового ICN-узла, требуется на этом ICN-узле установить связь с любым из SDN-коммутаторов.

Для моделирования также используется виртуальные машины под управлением ОС Ubuntu версии 14.04.1 с предустановленными компонентами:

1) Open vSwitch 2.0.2
2) java default-jdk
3) NDNFlow [18]
4) ccnx [19]
5) pox [20]
6) ccnping [21]

6. Экспериментальное исследование

6.1. Цель и методика исследования

Целью исследования является получение заключения относительно достоверности обоснования гипотезы.

Для проведения исследования используется следующая методика.

Исследование проводится путем имитационного моделирования.

Используемая имитационная модель позволяет создать ICN сеть с заданной топологией и запустить на ее узлах, соединенных в пары в

соответствии с заданной топологией, необходимое для проведения эксперимента программное обеспечение (Потребителя и Поставщика).

На каждом из ICN-узлов находится Поставщик, который удовлетворяет Интересы, приходящие от Потребителей, которые также находятся на каждом из ICN-узлов. Потребители отправляют Интересы и ждут пока Поставщик их удовлетворит. Предполагается, что в ICN-сети есть только один Поставщик для каждого экземпляра данных - то есть данные попарно различны.

Реакция ICN-узла при получении Интереса бывает двух типов: с отправлением Данных или с перенаправлением Интереса.

Размеры стенда, на котором проводится имитационное моделирование, ограничены снизу адекватностью модели, а сверху - объемом доступных ресурсов.

Измерение коэффициентов утилизации ресурсов ICN сетей осуществляется с помощью консольных утилит ccnping и ccnxHelloWorld, посредством наблюдения трафика на сетевых интерфейсах виртуальных машин, на которых располагаются ICN-узлы.

6.2. Описание экспериментальных данных

В силу того, что в ICN-сети в данном исследовании не используется кэширование и поскольку алгоритм маршрутизации сообщений детерминирован, то достаточно одного запуска Потребителя с каждого ICN-узла, при условии полного перебора всех остальных узлов в качестве ICN-узлов, на которых запущен Поставщик.

Таким образом, происходит измерение количества ICN-узлов, задействованных для реализации запроса, для каждой пары вершин в графе, представляющем заданную сеть, при этом одна из вершин пары соответствует Потребителю, а другая - Поставщику.

Цель - рассчитать отношение коэффициентов утилизации ресурсов ICN сети, построенной с помощью технологий SDN сетей, и ICN сети, построенной с помощью технологий традиционных сетей.

6.3. Результат экспериментов

Результат экспериментов представлен в Таблице 1, где в первом столбце указаны номера ICN-узлов, на которых в соответствующем эксперименте располагается Потребитель, а в первой строке - номера ICN-узлов, на которых в соответствующем эксперименте располагается Поставщик. В ячейках Таблицы 1 на пересечении строки соответствующего Потребителя и столбца соответствующего Поставщика расположено значение отношения коэффициентов утилизации ресурсов ICN сети, построенной с помощью технологий SDN сетей, и ICN сети, построенной с помощью технологий традиционных сетей.

Таблица 1

Результат экспериментов

№ узла	1	2	3	4	5	6	7	8	9	0
1	0.1	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4
2	0.2	0.1	0.3	0.2	0.2	0.4	0.3	0.3	0.4	0.4
3	0.2	0.3	0.1	0.2	0.3	0.2	0.3	0.4	0.3	0.4
4	0.2	0.2	0.2	0.1	0.2	0.3	0.2	0.3	0.3	0.3
5	0.3	0.2	0.3	0.2	0.1	0.4	0.3	0.2	0.4	0.3
6	0.3	0.4	0.2	0.3	0.4	0.1	0.2	0.3	0.2	0.3
7	0.3	0.3	0.3	0.2	0.3	0.2	0.1	0.2	0.2	0.2
8	0.4	0.3	0.4	0.3	0.2	0.3	0.2	0.1	0.3	0.2
9	0.4	0.4	0.3	0.3	0.4	0.2	0.2	0.3	0.1	0.2
0	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.1

Как видно из таблицы, во всех случаях отношение коэффициентов утилизации ресурсов ICN сетей меньше единицы, поэтому цель исследования считается достигнутой. Кроме того, важен тот факт, что с ростом числа хопов между Потребителем и Поставщиком, данное соотношение также увеличивается (пропорционально числу хопов).

6.4. Результат исследования

В результате исследования было проведено 100 экспериментов для случая организации ICN сети с использованием технологий SDN сетей, каждый из которых показал, что и Интерес, и Данные наблюдались только на ICN-узлах, составляющих кратчайший путь p, описанный в гипотезе; и 100 экспериментов для случая организации ICN сети с использованием технологий традиционных сетей, каждый из которых показал, что каждый отправленный Интерес приходил на множества узлов мощностью V и N, описанные в гипотезе, а Данные наблюдались на множестве узлов V, также описанном в гипотезе.

Соответствие между экспериментами позволяет сделать вывод о достоверности обоснования гипотезы о том, что ICN сети, построенные с помощью технологий SDN сетей, более эффективны по сравнению с ICN сетями, построенными с помощью технологий традиционных сетей.

7. Заключение

В данной статье описаны следующие основные полученные результаты и выводы проведенной исследовательской работы:

1) Изучена предметная область и подходы к построению ICN.

2) Обоснована гипотеза о том, что ICN сети, построенные с помощью технологий SDN сетей, более эффективны по сравнению с ICN сетями, построенными с помощью технологий традиционных сетей.

3) Проведено экспериментальное исследование, позволяющее сделать вывод о достоверности обоснования гипотезы.

Литература

1. ICN [HTML]
(https://en.wikipedia.org/wiki/Information-centric_networking)
2. SDN [HTML]
(https://en.wikipedia.org/wiki/Software-defined_networking)
3. Ilya Moiseenko, Lijing Wang, Lixia Zhang
Consumer / Producer Communication with Application Level Framing in Named Data Networking [PDF] 2016
(http://named-data.net/wp-content/uploads/2016/01/consumer_producer_communication.pdf)
4. Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang Named Data Networking [PDF] 2014
(http://named-data.net/wp-content/uploads/2014/10/named_data_networking_ccr.pdf)
5. Р.Л. Смелянский Программно-конфигурируемые сети - Новый подход к построению компьютерных сетей [HTML] 2012
(http://www.osp.ru/os/2012/09/13032491/)
6. Niels L. M. van Adrichem, Fernando A. Kuipers
NDNFlow: Software-Defined Named Data Networking [PDF] 2015
(https://www.nas.ewi.tudelft.nl/people/Fernando/papers/NDNflow.pdf)
7. Ali Ghodsi, Scott Shenker, Teemu Koponen, Ankit Singla, Barath Raghavan, James Wilcox Information-Centric Networking: Seeing the Forest for the Trees [PDF] 2011
(http://www.eecs.berkeley.edu/~alig/papers/information-centric-networking-seeing-the-forest-for-the-trees.pdf)
8. Parc CCNx 1.0 Evolution From Experiments Computer Science Laboratory Networking & Distributed Systems [PDF] 2014
(http://www.ccnx.org/pubs/hhg/1.7%20CCNx%201.0%20Evolution%20From%20Experiments.pdf)
9. Marc Mosko, Ignacio Solis, Ersin Uzun, Christopher Wood CCNx 1.0 Protocol Architecture [PDF] 2015
(http://www.ccnx.org/pubs/CCNxProtocolArchitecture.pdf)
10. Dimitris Syrivelis, George Parisis, Dirk Trossen, Paris Flegkas, Vasilis Sourlas, Thanasis Korakis, Leandros Tassiulas Pursuing a Software Defined Information-Centric Network [PDF] 2012
(http://nitlab.inf.uth.gr/NITlab/papers/pursuit_sdn_final.pdf)
11. Boyang Zhou, Chunming Wu, Xiaoyan Hong, Ming Jiang Algorithms for Distributed Programmable Controllers [PDF] 2013
(http://hong.cs.ua.edu/DCP-techReport-March2013.pdf)

12. Christos Tsilopoulos, George Xylomenos, Yannis Thomas Reducing Forwarding State in Content-Centric Networks with Semi-Stateless Forwarding [PDF] 2014
(http://mm.aueb.gr/publications/2014-IBF-INFOCOM.pdf)
13. ESXi [HTML]
(http://www.vmware.com/ru/products/vsphere-hypervisor.html)
14. CCNx_Distillery [HTML]
(https://github.com/PARC/CCNx_Distillery)
15 ccnxHelloWorld [HTML]
(https://github.com/PARC/ccnxHelloWorld)
16. ccnxWireshark [HTML]
(https://github.com/PARC/ccnxWireshark
17. Openflow [HTML]
(https://www.opennetworking.org/Openflow)
18. SDN-NDNFlow [HTML]
(http://github.com/TUDelftNAS/SDN-NDNFlow)
19. ccnx [HTML]
(https://github.com/ProjectCCNx/ccnx)
20. pox [HTML]
(https://github.com/noxrepo/pox)
21. ccnping [HTML]
(https://github.com/NDN-Routing/ccnping)

Раздел II.
Облачные вычисления

Пинаева Н.М., Антоненко В.А.

Разработка и реализация системы управления виртуальными сетевыми функциями в облачной платформе[1]

Введение

Современные телекоммуникационные сети содержат большое количество проприетарного оборудования, как правило, узкоспециализированного, созданного для реализации конкретной функциональности. Внедрение новой услуги требует установки нового комплекта оборудования, поддерживающего необходимую функциональность и его настройки. Помимо высоких капитальных и операционных затрат, такой подход инертен, не предоставляет возможностей динамического масштабирования, автоматической настройки сервиса и эффективного использования ресурсов. Решить данные проблемы призван принцип виртуализации сетевых функций.

Виртуализация сетевых функций (Network Functions Virtualization, NFV) — это технологии разделения логики сетевых функций и оборудования, на котором они выполняются, с помощью технологии виртуализации физических ресурсов.[1]

Виртуализация на базе технологии NFV строится на понятии виртуальной сетевой функции. Данное понятие формируется из следующих определений:

- Сетевая функция (Network Function, NF) — функциональный блок в сетевой инфраструктуре с четко определенными внешними интерфейсами и функционированием.
- 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.

[1] Проект поддержан грантом Инновационного фонда «Сколково» от 2 июля 2012г. №79

- Виртуальная сетевая функция (Virtualised Network Function, VNF) — реализация сетевой функции, которая может быть установлена и работать в инфраструктуре NFVI.
- Сетевой сервис (Network Service, NS) — композиция сетевых функций, определяемых своей функциональной и поведенческой спецификацией.

Для управления новыми сущностями, соответствующими введенным понятиям, определяются специальные системы, т.н. функциональные блоки:

- Менеджер виртуальной инфраструктуры (Virtualised Infrastructure Manager, VIM) — функциональный блок, обеспечивающий контроль ресурсов NFVI и управление ими, чаще всего отвечающий за один домен NFVI.
- Менеджер виртуальных сетевых функций (Virtualised Network Function Manager, VNFM) — функциональный блок, обеспечивающий управление жизненным циклом VNF.
- Оркестратор виртуализации сетевых функций (Network Functions Virtualisation Orchestrator, NFVO) — функциональный блок, обеспечивающий управление жизненным циклом сетевых сервисов, координацию управления жизненным циклом сетевых сервисов, VNF с помощью VNFM и NFVI с помощью VIM для обеспечения оптимального выделения необходимых ресурсов и их взаимосвязи.

Структурный шаблон управления и оркестрации для виртуализации сетевых функций (NFV Managment And Orchestration, NFV MANO) [2], разработанный группой Европейского Института Телекоммуникационных Стандартов (ETSI), обеспечивает управление инфраструктурой NVFI и оркестрацию выделения ресурсов, необходимых для создания сетевых сервисов и VNF. Подобная функциональность необходима для отделения сетевых функций от инфраструктуры NFVI.

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 был создан язык спецификаций TOSCA (Topology and Orchestration Specification for Cloud Applications) [3] — язык спецификаций, который позволяет описывать структуру облачных приложений и аспекты управления им. Описание функции с использованием данного стандарта называется TOSCA-шаблоном. Существует спецификация этого стандарта, которая позволяет описывать TOSCA-шаблоны функций на удобном для восприятия человеком языке YAML [4]. Все необходимые данные для установки и функционирования VNF, включая TOSCA-шаблон и, возможно, некоторые скрипты для управления функцией, образ операционной

системы, в которой должно работать приложение, реализующее VNF и др., объединяются в архив, называемый VNF-пакетом. Таким образом, для предоставления некоторой новой VNF необходимо сформировать VNF пакет и предоставить VNFM доступ к нему. Целью данной работы анализ результатов функционального исследования и демонстрация возможностей ПО поддержки жизненного цикла виртуальных сетевых функций на базе платформы C2, исходя из следующих требований:

- Должна присутствовать возможность управлять каждым этапом жизненного цикла VNF;
- Должен быть реализован интерфейс с VIM (запрос ресурсов через NFVO, обеспечивает независимость VNFM от VIM);
- описание VNF на спецификации должно быть представлено TOSCA YAML с использованием основных типов узлов: SoftwareComponent, Compute [5];
- Обеспечение мониторинга VNF-специфичных параметров (под VNF-специфичными параметрами подразумеваются некоторые параметры, характеризующие работу VNF).

2. Схожие работы

На данный момент существует несколько реализаций системы управления VNF. Ниже рассмотрены решения, наиболее подходящие под требования, описанные во введении.

1. Cloudify

Cloudify [6] — открытая программная платформа для оркестрации облачных технологий, использующая TOSCA. В ней автоматизированы установка и размещение функций, мониторинг и восстановление после возникновения ошибок. Cloudify работает как программный исполнитель, который реализует все обыкновенные шаги жизненного цикла приложений полностью автоматически. Для достижения такого уровня автоматизации Cloudify принимает на вход конфигурационные файлы приложения, называемые blueprints, которые описывают взаимодействие приложения с дата-центром через установленные API.

Основой архитектуры Cloudify является модуль, объединяющий NFVO и VNFM, таким образом, данная архитектура не соответствует стандарту ETSI, не позволяет использовать несколько NFVM с одним NFVO, более того, данная интеграция создает зависимость NFVM от VIM, т.к. архитектура системы подразумевает возможность занятия ресурсов самим менеджером.

2. OpenStack Tacker

Объединением OpenStack был запущен OpenStackTacker [7] — проект для управления жизненным циклом сетевых сервисов. Tacker исполняет роль VNFM: его архитектура позволяет управлять

экземплярами VNF с использованием компонентов OpenStack: Nova, Glance и Neutron. Tacker принимает на вход TOSCA-шаблоны, содержащие дескрипторы виртуальной сетевой функции.

Таким образом, главным недостатком данного решения является жесткая привязка VNFM к сервисам OpenStack.

3. OpenBaton

OpenBaton [8] — проект с открытым исходным кодом, реализующий архитектуру ETSI MANO. Модуль, отвечающий за управление виртуальными сетевыми функциями, VNFM, управляет жизненным циклом VNF в соответствии с их дескрипторами.

VNFM реализует соответствующий модуль ETSI MANO архитектуры. Он является промежуточным компонентом между NFVO и виртуальными сетевыми функциями, в частности, виртуальными машинами, на которых выполняется программное обеспечение VNF. Для управления жизненным циклом VNF VNFM взаимодействует с системой EMS (Element Management System), работающей в качестве агента на виртуальных машинах и запускающей скрипты, содержащиеся в VNF пакете.

Основным недостатком данного проекта является формат принимаемого шаблона функции, описывающего аспекты установки функции и управления ей. А именно, шаблон должен быть написан на разработанном OpenBaton языке описания. Данным способом можно описать значительно меньше параметров функции и особенностей управления ей, чем с использованием TOSCA, и разработанный язык не позволяет вносить пользовательские расширения самого языка описания, в отличие от TOSCA.

Таким образом, на данный момент системы управления VNF, отвечающей поставленным требованиям не существует.

3. Функциональность и внутреннее устройство основных блоков VNFM

В ходе работы был предложен модуль VNFM, реализующий систему управления виртуальными сетевыми функциями.

Разработанный модуль VNFM состоит из нескольких функциональных блоков, каждый из которых имеет свои задачи:

- Блок для работы с базой данных (Database)
- Блок для отправки сообщений другим модулям и их получения (Messenger)
- Блок для работы с TOSCA-шаблонами (Tosca)

1. Database

Так как VNFM управляет жизненным циклом экземпляров VNF, ему необходимо хранить некоторую информацию о работающих экземплярах VNF и загруженных VNF пакетах, из которых VNFM

извлекает необходимую информацию для запуска экземпляра функции и управления им.

Функциональный блок Database предназначен для работы с базой данных, в которой хранится необходимая информация об экземплярах VNF, которыми управляет данный менеджер. Для хранения данных используется MySQL. Выбор MySQL связан с поддержкой совместимости с наиболее популярным на данный момент базовым ПО для реализации VIM – OpenStack.

2. Messenger

Так как VNFM является частью шаблона NFV MANO, ему нужны интерфейсы с другими модулями. В данной реализации предполагается, что модули обмениваются сообщениями, таким образом координируя свою работу.

Были реализованы следующие основные интерфейсы:

- С оркестратором
  - register_vnf (добавление нового VNF пакета)
  - create_vnfi (создание экземпляра функции)
  - update_tenant (обновление виртуальной инфраструктуры)
- С мониторингом
  - add_monitor (обеспечение мониторинга некоторых параметров)
  - recovery_vm (достижение критического значения некоторого параметра)

Функциональный блок Messenger предназначен для получения, отправки и обработки сообщений. Обмен сообщений реализован с помощью RabbitMQ [9].

3. Tosca

Функциональность VNFM в большой мере зависит от возможностей его TOSCA-интерпретатора (интерпретатора TOSCA-шаблонов), так как именно TOSCA-шаблон определяет поведение менеджера и особенности управления каждой VNF.

Функциональный блок Tosca предназначен для работы с TOSCA-шаблонами и включает в себя TOSCA-интерпретатор, переводящий шаблон во внутреннее представление и собственно интерпретатор, выполняющий основные функции, связанные с управлением жизненным циклом экземпляра VNF.

4. Функциональное тестирование

Для проверки функционирования и соответствия поставленным требованиям модуля VNFM проведено функциональное тестирование, состоящее из следующих экспериментов:

- Создание экземпляра функции — проверка интерпретации TOSCA-шаблона, порядка установки узлов TOSCA-шаблона, выполнения планов управления для каждого из узлов TOSCA-шаблона (сначала create, затем configure), установки параметров мониторинга;
- Автоматическое масштабирование экземпляра функции при достижении граничного значения заданного параметра;
- Восстановление работы экземпляра функции при обнаружении некорректной работы (недостижимость виртуальной машины, на которой выполняется экземпляр VNF)

Все вышеперечисленные эксперименты проводятся для VNF squid, в VNF-пакет (см. Введение) которой входит:

- TOSCA-шаблон на языке YAML
- скрипт squid_install.sh
- скрипт squid_configure.sh

В TOSCA-шаблоне данной VNF содержится следующая информация:

- Шаблон топологии состоит из двух узлов: proxy типа SoftwareComponent и server типа Compute и связи между этими узлами типа hosted_on;
- Для узла server определены требования к виртуальной машине;
- Для узла squid определены стандартные операции жизненного цикла: create с ссылкой на скрипт squid_install.sh и configure с ссылкой на скрипт squid_configure.sh;
- Правило для мониторинга: загрузка процессора не должна превышать 60%;
- Правило для масштабирования: добавление дополнительного server и распределение нагрузки между всеми узлами при достижении граничного значения заданного параметра для мониторинга.

Для получения результатов экспериментов реализована возможность просмотра полученных и отправленных сообщений в модулях NFVO, VIM, NFVM.

В базе данных можно видеть результаты регистрации VNF (рис.1) и создания экземпляра функции с соответствующей топологией (рис.2, 3).

Рис. 1. Зарегистрированная VNF squid

Рис. 2. Экземпляр функции squid

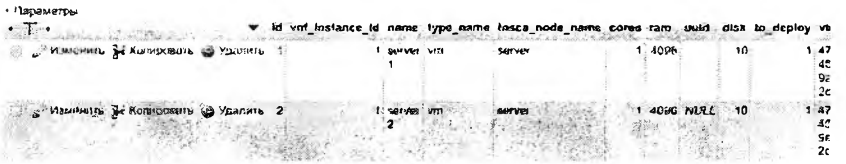

Рис. 3. Виртуальная машина созданного экземпляра функции squid

В результате масштабирования количество виртуальных машин функции увеличилось на 1 (рис.4).

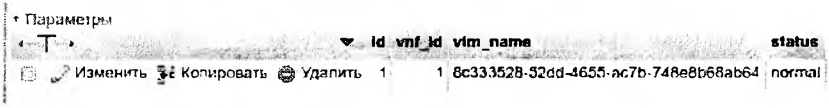

Рис. 4. Результат масштабирования экземпляра функции squid

Функциональное тестирование показало корректную работу разработанного модуля в основных вариантах использования, соответствующую поставленным требованиям, и правильность взаимодействия с другими модулями, обеспечивающими необходимую функциональность.

Заключение

В ходе работы была разработана и реализована система управления виртуальными сетевыми функциями в облачной платформе. Предложенная система не зависит от функциональных блоков нижнего уровня и принимает на вход VNF шаблоны, написанные на наиболее распространенном, поддерживающем пользовательские расширения и удобном для описания VNF языке TOSCA.

Проведено функциональное тестирование данного модуля в основных вариантах использования: создание экземпляра VNF, автоматическое масштабирование экземпляра функции, восстановление работы экземпляра функции при возникновении ошибки.

Планируется расширение функциональности модуля для поддержки большего количества типов TOSCA узлов и развитие системы мониторинга.

Литература

1. Network functions Virtualisation (NFV); Terminology for Main Concepts in NFV [PDF] (http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_NFV003v010201p.pdf)

2. Network functions Virtualisation (NFV); Management and Orchestration [PDF] (http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf)

3. Topology and Orchestration Specification for Cloud Applications Version 1.0; OASIS Standard [HTML] (http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html)

4. The Official YAML Web Site [HTML] (http://yaml.org/)

5. TOSCA Simple Profile in YAML Version 1.0 [HTML] (http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csprd01/TOSCA-Simple-Profile-YAML-v1.0-csprd01.html)

6. Cloudify [HTML] (http://getcloudify.org/about_us.html)

7. Tacker — OpenStack NFV Orchestration [HTML] https://wiki.openstack.org/wiki/Tacker

8. OpenBaton [HTML] (http://openbaton.github.io)

9. RabbitMQ — Messaging that just works [HTML] (https://www.rabbitmq.com/)

Кочетков П.А., Антоненко В.А.

Организация сетевого взаимодействия между компонентами систем NPS[1]

Введение

Проектирование сетевой инфраструктуры является весьма трудоемкой задачей. Зачастую, сети подвержены сбоям из-за большой сложности и размера. Кроме того, вмешательство в инфраструктуру с целью диагностики и отладки само по себе может послужить источником дополнительных рисков и угроз нормальному режиму работы ИТ-комплекса. Поэтому, оптимальным решением данной проблемы является анализ поведения системы с помощью моделирования. Моделирование позволяет оценивать эффективность сетевой топологии, поведение и параметры сети, проектировать и отлаживать работу сетевой инфраструктуры предприятий и отрабатывать механизмы отказоустойчивости.

Система имитационного моделирования (СИМ) - комплекс программных средств для создания имитационной модели и ее симуляции. Одним из существенных недостатков СИМ является необходимость доказывать адекватность и корректность построенной модели. Однако, используя подход HI-FI (High-Fidelity), подразумевающий использование легковесных контейнеров в качестве хостов, можно избежать этих доказательств, поскольку он обладает точностью, соответствующей точности реального оборудования.

Система NPS (Network Prototyping Simulator) [1] является HI-FI СИМ компьютерной сетей, поддерживающей распределенное функционирование, то есть работу на нескольких вычислителях (узлах). NPS обладает удобной графической средой разработки, богатым набором функций и горизонтальной масштабируемостью.

Чтобы обеспечить масштабируемость, NPS делит граф топологии сети моделируемой сети на подграфы (по умолчанию в равных пропорциях с точки зрения количества вершин) и производит моделирования каждого подграфа на отдельном NPS-узле. В текущей версии системы, деление графа происходит таким образом, чтобы сократить количество связей между подграфами до одного ребра. Как следствие, вычислительные ресурсы могут быть загружены неравномерно.

Система NPS представляет стенд, состоящий из серверов, соединённых физическими каналами. При этом пользователи лишены

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

возможности задавать каналам характеристики, такие как пропускная способность и задержка, хотя эта функциональность является крайне важной для пользователей СИМ. Моделируемые каналы прокладываются поверх физических. Характеристики этих каналов не обязаны совпадать. Более того может оказаться так, что характеристики физ. Канала окажутся ниже характеристик логического.

Таким образом, задача организации сетевого взаимодействия между узлами NPS посредством виртуальных каналов является важной и актуальной.

Данная статья организации сетевого взаимодействия в системе NPS между компонентами системы (NPS узлами). Причем, вариант этого взаимодействия должен решать следующие проблемы: неравномерную загрузку вычислительных ресурсов, а также поддерживать возможность задания пропускной способности и задержки логических каналов.

Оставшаяся часть статьи будет организована следующим образом: сначала, в главах 1 и 2 будет описано устройство системы NPS и основные принципы ее функционирования, рассмотрено, как именно в настоящий момент организовано взаимодействие между узлами NPS, к каким ограничениям такая организация приводит, а также предложен вариант решения этих проблем. Затем, во главе 3, будет проведен обзор подходов к организации сетевого взаимодействия между компонентами системы NPS, а также будет выбран наиболее подходящий. Глава 4 посвящена реализации выбранного подхода, в ней приведены используемые библиотеки и утилиты, а также некоторые важные моменты программной реализации. В конце будет сделано заключение по результатам работы.

1. Описание системы NPS

Как было описано во введении, NPS – HI-FI система имитационного моделирования. В основе HI-FI систем лежит так называемая легковесная виртуализация – метод виртуализации на уровне операционной системы, позволяющий запускать несколько изолированных экземпляров системы Linux (контейнеров) на одном хосте управления с использованием одного ядра Linux. Более подробно об используемой в NPS легковесной виртуализации можно узнать из статьи [2].

Легковесная виртуализация для моделирования впервые была применена в Mininet - системе прототипирования компьютерной сети с открытым исходным кодом, разработанной в Стэнфордском университете. Методы, используемые в данном подходе, позволяют детально моделировать процесс функционирования сети: воспроизводить процессы обработки и передачи сетевого трафика. Функционирование сети задается настройками виртуального сетевого

стека операционной системы машины, на которой осуществляется процесс моделирования, что фактически означает использование реального сетевого стека [3]. Это позволяет свести к минимуму задачу доказательства адекватности и корректности модели, которая, зачастую, является более трудоемкой, чем само моделирование. Более подробно о системе Mininet можно узнать из статьи [4].

Несмотря на высокую точность и эффективность, Mininet применим только для моделирования сетей малого размера, топология которых состоит не более чем из 2000 узлов [3], в связи со следующими ограничениями [4]:

- один сервер, работающий в режиме реального времени, неизбежно будет ограничивать пропускную способность и число процессов
- большое количество TCP потоков увеличивает общую нагрузку на процессор

Эти ограничения делают невозможным использование Mininet для моделирования глобальной компьютерной сети - сети, состоящей из 10^5-10^6 хостов. Однако решение этой проблемы может быть достигнуто за счет поддержки распределенного функционирования, т.е. работы на нескольких вычислителях [5].

NPS является реализацией этой идеи - создать некоторую надстройку над Mininet, которая позволяет объединить несколько отдельных Mininet-экземпляров в один кластер [6]. Сетевая топология, заданная пользователем, разбивается на несколько частей, и каждая из этих частей моделируется отдельным экземпляром Mininet, на отдельной физической машине. За счет этого, удается добиться одного из важнейших преимуществ системы NPS - горизонтальной масштабируемости: увеличение максимального количество моделируемых хостов можно достичь за добавления новых узлов кластера.

Таким образом, NPS обладает всеми теми же преимуществами, что и Mininet, но кроме этого, позволяет моделировать глобальные компьютерные сети.

Более подробно об устройстве, принципах и архитектуре системы NPS можно узнать из [3].

2. Организация взаимодействия между компонентами системы NPS

Итак, в рассматриваемом варианте, NPS представляет из себя сеть из реальных машин, на каждой из которых Mininet моделирует определенный участок сети. Упрощенно, процесс моделирование сети в системе NPS происходит следующим образом:

1. Пользователь задает сетевую топологию (рис. 1)

2. Система NPS делит сетевой граф на подграфы
3. Каждый подграф моделируется с помощью Mininet на отдельном физическом вычислителе. При этом, все вычислители объединены сетью (рис. 2)

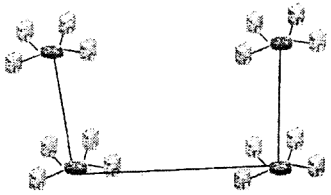

Рис. 1. Сетевая топология, заданная пользователем

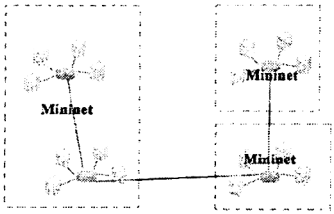

Рис. 2. NPS-узлы, соединенные сетью и моделирующие отдельные подграфы заданной топологии, изображенной на рис. 1

Как можно заметить из рис. 1, некоторые каналы моделируемой сети накладываются на физические каналы между узлами. Разбиение графа моделируемой сети можно осуществлять на основании нескольких критериев:

1. Делить граф сети на подграфы таким образом, чтобы между NPS-узлами был только один моделируемый канал (данный функциональной доступна путем изменения опцией модуля разбиения графа топологии сети на подграфы), который накладывается на физический. Это решит проблему изоляции трафика, но, во-первых, такое разделение не всегда возможно (например, полносвязный граф), а во-вторых, часто, такое разделение может привести к нерациональному использованию ресурсов: когда один NPS-узел будет сильно перегружен, а другие практически простаивать. На текущий момент, именно такая реализация используется в NPS.

2. Делить граф на подграфы таким образом, чтобы обеспечить сбалансированную загруженность NPS-узлов, например, так, чтобы каждый NPS-узел моделировал одинаковое количество хостов (данном контексте считается, что нагрузка сетевых приложений, функционирующих на хостах, распределена равномерно между вершинами графа топологии сети). Однако, при этом, может сложиться ситуация, когда сразу несколько каналов моделируемой сети наложатся на один физический канал, соединяющий NPS-узлы (рис. 3). При таком варианте разделения, особо остро встанет вопрос изоляции трафика и контроля качества сервиса.

Вариант разделения, описанный в пункте 2, представляется наиболее правильным, и именно его реализация является целью курсового проекта. Однако, такое разделение сопряжено с описанными выше проблемами.

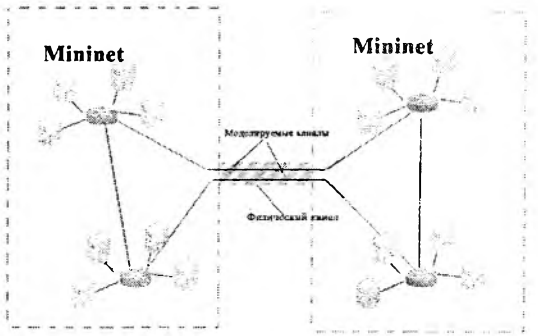

Рис. 3. Несколько каналов моделируемой сети, ассоциированные с одним физическим каналом

Решением этих проблем является использования техники виртуализации каналов для моделирования нескольких логических каналов на одном физическом. Использование виртуальных каналов позволит не только разделить трафик, но и предоставит возможность регулировать характеристики моделируемого канала - задержку и пропускную способность.

3. Обзор подходов к организации взаимодействия между компонентами системы NPS

В данной разделе будет проведен обзор технологий виртуализации каналов с целью определения базового подхода для решения поставленной задачи.

Для сравнения было выбрано три техники виртуализации: VLAN, VxLAN, GRE-Tunneling. Ознакомиться с этими технологиями можно в статьях [7], [8], [9], соответственно. Критерии и их обоснование будут приведены по ходу обзора.

Изоляция по пропускной способности канала. Под изоляцией будем понимать возможность независимого управления шириной пропускания виртуальных каналов, размещенных поверх физического канала. Изоляция по пропускной способности, пожалуй, является самым важным пунктом данного обзора. Техника виртуализации, не способная обеспечить это требования, не нуждалась бы в дальнейшем

рассмотрении. Модель сети, построенная на каналах без изоляции по пропускной способности, не являлась бы корректной.

Все три рассматриваемые техники виртуализации позволяют производить изоляцию по пропускной способности.

Номинальная задержка канала. Регулировка номинальной задержки канала является крайне важным аспектом моделирования сети. Такие задержки практически всегда присутствуют в реальных сетях. Они могут устанавливаться как сетевым администратором как параметр сети, так и появляться в процессе работы сети из-за нагрузок или выхода из строя оборудования. Кроме того, бывает полезно оценить поведение сети при изменении этого параметра (например, при проведении нагрузочного тестирования). Поэтому предоставление возможности регулировать этот параметр в моделируемой сети является важным для СИМ.

Все три рассматриваемые техники виртуализации позволяют выставлять номинальную задержку канала.

Совместимость с другими системами виртуализации. В настоящее время, реальные сетевые инфраструктуры предприятий практически всегда имеют собственные виртуальные каналы, например, для разделения трафика между отделами и подразделениями компании. Поэтому, важно предоставить пользователям СИМ возможность создавать эти виртуальные каналы и в модели сети.

Для обеспечения этого, выбранная технология виртуализации не должна вступать в конфликт с задаваемой пользователем виртуализацией.

Технология VLAN, в силу работы на канальном уровне, не в состоянии обеспечить совместимость с другими системами виртуализации. Например, невозможно создать VLAN поверх VLAN, или GRE поверх VLAN.

Техники же GRE и VxLAN такими ограничениями не обладают и совместимы с другими системами виртуализации.

Сложность настройки виртуального канала. Важно отметить, что в соответствии с архитектурой NPS, NPS-узлы – это всегда машины с установленной на них операционной системой Ubuntu 14.04. Поэтому, важным требованием к простоте настройки является наличие утилиты в ОС Linux, поддерживающей прокладывание соответствующего виртуального канала.

Утилита ip-link, являющаяся частью пакета iproute2, позволяет выполнять прокладывание VLAN, VxLAN и GRE-тунелей.

Однако, с точки зрения простоты, документированности команд, а также распространенности различных инструкций по настройке и обсуждаемости на форумах, наиболее удобным оказывается GRE. Вслед за ним VLAN, а уже потом VxLAN.

Стоит отметить, в системе Maxinet [10], которая является аналогом системы NPS, для решения аналогичной задачи была использована виртуализация каналов с помощью GRE-тунелей.

Поддержка QoS (policing & shaping). Shaping & Policing трафика позволяет придавать профилям потоков необходимую форму с помощью встроенных средств коммутатора. Если интенсивность потока превышает спецификации профиля, то shaping задерживает обработку пакетов, policing сбрасывает пакеты.

Тестирование QoS является крайне важным этапом моделирования компьютерной сети и обязательно должно поддерживаться системой.

Все три рассматриваемые техники виртуализации поддерживают QoS.

Таблица 1

Сравнение технологий виртуализации каналов

	VLAN	VxLAN	GRE
Изоляция по пропускной способности канала	+	+	+
Установление номинальной задержки канала	+	+	+
Совместимость с другими системами виртуализации каналов	-	+	+
Сложность настройки виртуального канала	+	-	+
Поддержка QoS	+	+	+

Как видно из табл. 1, исходя из проведенного обзора, наиболее подходящим для использования в системе NPS выглядит виртуализация посредством GRE-туннелей.

4. Реализация

Система NPS написана на языке Python, поэтому, именно этот язык и был выбран для реализации.

Следует еще раз отметить алгоритм работы системы NPS:

1. Пользователь задает сетевой граф
2. Контрольный узел делит сетевой граф на подграфы
3. Для каждого подграфа контрольный узел генерирует скрипт, который запускает Mininet с заданной топологией
4. Скрипты рассылаются по SSH и выполняются. Производится моделирование

Таким образом, для решения поставленной задачи с практической точки зрения необходимо:

- Модифицировать работу алгоритма деления сетевого графа на подграфы
- Модифицировать процедуру генерации скриптов, добавив прокладывание GRE-туннелей
- Разрешить возникающие архитектурные коллизии
- Добавить поддержку задания номинальной задержки и пропускной способности каждого туннеля

Для того чтобы разбить сетевой граф на равные по количеству узлов подграфы, использовалась библиотека Metis, которая предоставляет широкую функциональность для работы с графами.

Для прогладки GRE-туннелей в ОС Linux наиболее удобной является утилита ip-link. Для этого, на узлах, между которыми нужно проложить туннель выполняются команды:

ip link add <GRE_NAME> type gretap local <IP1> remote <IP2>

ip link set <GRE_NAME> up

Разработанный модуль, получив данные о том, как именно разбит сетевой граф, генерирует список пар ip-адресов для прокладки GRE-туннелей. После чего, скрипт, который будет рассылаться на узлы NPS, модифицируется: в него добавляется выполнение описанных выше Linux-команд для поднятия GRE-туннелей. Затем, эти туннели ассоциируется с соответствующими моделируемыми каналами средствами Mininet.

Отдельной задачей является обеспечение возможности устанавливать номинальную задержку и пропускную способность. Для ее решения использовалась утилита tc из пакета iproute, которая позволяет управлять очередями коммутации.

Для эмуляции задержки в канале, был использован модуль netem утилиты tc, который предоставляет функционал для эмуляция задержки, с различной функцией распределения. В данном случае, была реализована постоянная задержка, но этот параметр может быть легко изменен при необходимости.

Для ограничения пропускной способности канала, с помощью той же утилиты tc, был использован алгоритм иерархического текущего ведра. Все параметры, задающие задержку и пропускную способность каналов, берутся из конфигурационного файла.

Заключение

В рамках данной работы были изучены принципы работы и устройство системы NPS, проведен обзор технологий виртуализации каналов и выбрана технология виртуализации каналов, наилучшим образом подходящая к данной задаче.

С практической точки зрения, был разработан модуль, организующий сетевое взаимодействие между узлами NPS посредством GRE-туннелей. Предусмотрен интерфейс для задания пропускной способности и номинальной задержки в канале. Корректность работы модуля была доказана экспериментально.

Среди дальнейших планов - внедрение разработанного модуля в обновленную версию системы NPS.

Литература

1. Vitaly A., Ruslan S., Andrey N. Large scale network simulation based on hi-fi approach // Proceedings of the 2014 Summer Computer Simulation Conference. — США, Монтерей, 2014. — P. 20–27.

2. Merkel D. Docker: lightweight linux containers for consistent development and deployment //Linux Journal. – 2014. – T. 2014. – №. 239. – P. 2.

3. Антоненко В.А. Разработка и исследование модели функционирования глобальной сети для анализа динамики распространения вредоносного программного обеспечения: дис. канд. физ.-мат. наук: 05.13.11: защищена 19.09.2014. — М., 2014. — 108 с.

4. de Oliveira R. L. S. et al. Using mininet for emulation and prototyping software-defined networks //Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on. – IEEE, 2014. – C. 1-6.

5. Handigol N. et al. Reproducible network experiments using container-based emulation //Proceedings of the 8th international conference on Emerging networking experiments and technologies. – ACM, 2012. – P. 253-264.

6. Antonenko V., Smelyanskiy R. Global network modelling based on mininet approach //Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. – ACM, 2013. – P. 145-146.

7. Виртуальные локальные сети (VLAN) // Национальном Открытом Университете «ИНТУИТ» // URL: http://www.intuit.ru/studies/courses/3591/833/lecture/14258 (дата обращения: 08.04.2016)

8. Что такое и как работает технология VXLAN для создания виртуальных сетей нового поколения для виртуальных машин VMware vSphere // vmgu.ru / URL: http://www.vmgu.ru/news/vmware-vxlan-for-vsphere (дата обращения: 12.04.2016)

9. GRE — пример настройки и описание // CiscoTips / URL: http://ciscotips.ru/gre (дата обращения: 15.04.2016)

10. Wette P.et al.Maxinet: Distributed emulation of software-defined networks //Networking Conference, 2014 IFIP. – IEEE, 2014. P.1-9.

Романов А.Р., Антоненко В.А.

Разработка системы обеспечения надежного и масштабируемого виртуального сетевого сервиса в облачной среде[1]

Введение

В современных сетях функционирует огромное количество сервисов: маршрутизация (routing), трансляция сетевых адресов (NAT), сетевой экран(firewall), туннелирование (VPN), проски-сервер и т.д.. Многие из них реализованы в виде программно-аппаратного решения, которое требует дополнительных усилий при горизонтальном масштабировании, либо требует вертикального масштабирования. При необходимости внедрения нового сервиса в сеть приходиться приобретать новое оборудование. Программно-аппаратные решения ограничивают возможность обновления ПО и зачастую содержать избыточное количество сервисов. Функции (составная часть сервиса), реализованные в составе отдельных сетевых узлов, зачастую плохо масштабируются, так как при увеличении нагрузки на сеть увеличивается число необходимых физических устройств. При обычных (не пиковых) нагрузках часть устройств простаивает. Следовательно, становится актуальным вопрос динамической масштабируемости сервиса в зависимости от его загрузки.

Одной из проблем современных сетей является зависимость от производителя аппаратных устройств. Оборудование разных производителей может конфликтовать между собой. Со временем производители перестают поддерживать устаревшие устройства.

Целью данной работы является разработка решения, которое управляет жизненным циклом виртуальных сетевых сервисов и обеспечивает их надежную работу и масштабируемость.

Таким образом, можно выделить ключевые проблемы организации работы сетевого сервиса:

- использование оборудования с избыточной функциональностью;
- расчет производительности сервиса исходя из максимально возможной нагрузки;
- простаивание оборудования в случае, если нагрузка не является пиковой;

[1] Проект поддержан грантом Инновационного фонда «Сколково» от 2 июля 2012г. №79.

- зависимость от производителя оборудования (техническое обслуживание и устаревание оборудования, невозможность модифицировать сервис без вмешательства производителя).

Концепция Виртуальных Сетевых Функций (Network Function Virtualization, NFV) призвана решить указанные выше проблемы. Она позволяет виртуализировать сетевые сервисы, которые на данный момент реализованы лишь на физических устройствах. Под виртуализацией сетевых сервисов понимается предоставление сетевых услуг в виде программного обеспечения, функционирующего на одной или нескольких связанных виртуальных машинах. NFV работает в рамках модели SaaS [1]. Свойства концепции NFV:

- горизонтальная масштабируемость – в зависимости от загруженности сервиса будет задействована та часть инфраструктуры, которая необходима для корректной работы сервиса;
- надежность – в случае сбоев в работе сервиса будут предприниматься действия по восстановлению его работы в автоматическом режиме;
- высокая скорость развертки сервиса – виртуализация позволяет быстро внедрять новые сервисы, а также развертывать уже существующие сервисы на новой инфраструктуре.

Концепция NFV отделяет программную составляющую сетевых функций от аппаратной (вычислительные и сетевые ресурсы). Такой подход подразумевает использование физической инфраструктуры, не зависящей от производителя, что требует стандартизации интерфейсов между различными компонентами системы. Описание NFV рассматривается на основе стандарта [2].

Основное понятие концепции NFV - это виртуальная сетевая функция (VNF). Она представляет собой описание требуемой инфраструктуры, требуемого программного обеспечения, параметров подключения пользователей к этой услуге и т.д. Так же существует понятие экземпляра виртуальной сетевой функции (VNF instance). Экземпляр VNF – это уже размещенная виртуальная инфраструктура, на которой функционирует программное обеспечение, присутствующее в описании функции. Таким образом, для каждой VNF существует единственное описание и множество ее экземпляров. Все экземпляры функции независимы друг от друга. В общем случае, они могут быть размещены в разных доменах (в разных ЦОД). Подробнее о виртуальных сетевых функциях и их свойствах можно узнать в стандарте [3].

Виртуальный сетевой сервис (VNS) – это некоторое множество связанных между собой виртуальных сетевых функций. Это конечная услуга, которая будет предоставляться клиентам. Концепция NFV предполагает внутреннее представление VNS как произвольное не пустое множество, состоящее из VNF.

Наиболее интересен случай цепочек виртуальных сетевых функций (VNF chaining), так как становится понятной схема работы такого сервиса. В этом случае можно считать каждую VNS как цепочку виртуальных сетевых функций. С математической точки зрения, VNS – это суперпозиция функций, составляющих виртуализированный сервис. Пусть x – это входящий трафик некоторого объема. Тогда результатом работы сервиса S, состоящего из последовательности функций f_1, f_2, f_3 будет трафик y, такой что:

$$y = f_3(f_2(f_1(x))) = S(x) \quad (1)$$

Разработкой высокоуровневой архитектуры ETSI NFV Management and Orchestration (ETSI NFV MANO) для NFV платформ занимается организация ETSI. Главной особенностью архитектуры, предложенной ETSI, является оптимальное использование инфраструктуры: она выделяется для каждой функции по запросу. Базовыми блоками, из которых строятся виртуальные сетевые сервисы (VNS), являются виртуальные сетевые функции (VNF). Платформа на базе ETSI NFV MANO должна уметь размещать VNF на подконтрольной инфраструктуре. В результате комбинирования блоков VNF получаются виртуальные сетевые сервисы, которыми пользуются клиенты платформы.

Как показано на рисунке 1, в ETSI NFV MANO имеется 3 основных модуля:

- Менеджер виртуальной инфраструктуры (Virtualized Infrastructure Manager, VIM). Модуль обеспечивает виртуализацию физической инфраструктуры в рамках одного домена. Под доменом понимается центр обработки данных (ЦОД). ETSI NFV MANO предполагает использование нескольких менеджеров инфраструктуры (по одному на каждый домен).
- Менеджер виртуальных сетевых функций (Virtual Network Function Manager, VNFM). Модуль отвечает за полный жизненный цикл виртуальных сетевых функций. Архитектура предполагает возможность наличия нескольких менеджеров функций.
- Оркестратор виртуальных сетевых сервисов (Network Function Virtualization Orchestrator, NFVO). Данный модуль решает две основные задачи:
  - оркестрация ресурсов между несколькими менеджерами инфраструктуры, резервация ресурсов;
  - управление жизненным циклом виртуальных сетевых сервисов.

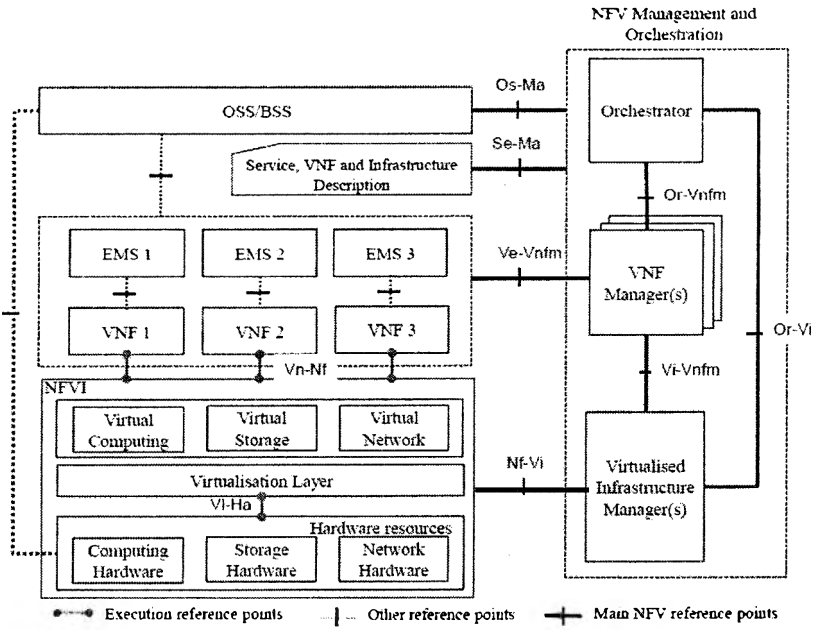

Рис. 1. Архитектура NFV Management and Orchestration

Каждый модуль обеспечивает определенный уровень абстракции. VIM занимается виртуализацией физических ресурсов. Менеджер функций VNFM предоставляет набор функций, размещенных на виртуальных ресурсах. Оркестратор NFVO управляет виртуальными сетевыми сервисами, которые построены на базе виртуальных сетевых функций.

Так же в архитектуре присутствуют неосновные модули:

- Описание виртуальных сетевых сервисов, функций и используемой ими инфраструктуры. В ETSI NFV MANO блоки, которые содержат такие описания, называют каталогами (catalog). В платформах, разрабатываемых на базе ETSI NFV MANO, функции каталогов обычно выполняют менеджеры соответствующего уровня (NFVO, VNFM, VIM).
- Система управления элементами (Element Management System, EMS).
- Система управляет работой элементов экземпляра виртуальной сетевой функции, отвечает за параметры функции. Данная система взаимодействует с менеджером функций через закрытые интерфейсы. Поэтому в существующих решениях, известных автору, данный модуль включен в состав менеджера функций.

Разрабатываемый модуль должен работать в облачной среде. Это означает, что все клиенты виртуальных сетевых сервисов являются виртуальными машинами. Далее под клиентом виртуального сетевого сервиса будем подразумевать виртуальную машину.

1. Обзор существующих NFV платформ

В рамках данной работы был проведен обзор существующих NFV платформ, с целью выявить достоинства и недостатки существующих решений. Решения будут сравниваться по следующим критериям:

- Соответствие архитектуры платформы стандарту ETSI NFV MANO.
- Независимость от платформы виртуализации ресурсов. Данный пункт означает, что решение использует программную прослойку (адаптер)для взаимодействия с платформой виртуализации. При необходимости использовать другую платформу достаточно заменить программную прослойку без переписывания кода основных модулей.
- Поддержка работы с несколькими VIM одновременно. Решение, обладающее данным свойством, способно управлять физическими ресурсами сразу нескольких ЦОД одновременно (один модуль VIM для каждого ЦОД).
- Мониторинг состояния виртуального сетевого сервиса. Поддержка этого свойства позволяет следить за состоянием инфраструктуры виртуальных сетевых сервисов.
- Автоматическое срабатывание обработчиков scaling и healing. Scaling – событие, связанное с масштабированием сервиса, healing – событие, связанное с некорректной работой сервиса. Решение, в котором реализован данный пункт, может автоматически запускать обработчики на события scaling и healing, чтобы восстановить работу сервиса. Описание обработчиков на события присутствуют в описании виртуальной сетевой функции.

Выполнение всех критериев, указанных выше, позволит решению выполнять задачи по управлению виртуальными сетевыми сервисами и обеспечивать их отказоустойчивость и масштабируемость в автоматическом режиме.

Результаты обзора, приведенные в таблице 1, показали, что ни одно из существующих решений полностью не удовлетворяет установленным требованиям [ссылка на статью Пинаевой].

Заметим, что для реализации автоматического восстановления сервиса необходимо наличие определений обработчиков в описании виртуальной сетевой функции. Ни одно из рассмотренных решений не имеет возможностей по анализу данного параметра. Однако такая возможность разрабатывается для платформы управления облачными

ресурсами C2 Platform. Поэтому данная платформа и была выбрана для реализации разработанной архитектуры и проведения экспериментов.

Таблица 1

Сравнение существующих NFV решений

Платформа	ETSI NFV MANO	Независимость от платформы виртуализации	Одновременная работа с несколькими VIM	Мониторинг состояния VNS	Автоматическое срабатывание обработчиков событий
OPNFV	+	+	-	?	?
Cloudify	+	+	?	+	-
Openstack Tacker	+	-	-	?	?
OpenBaton	+	+	+	+	+

В результате анализа существующих NFV платформ были сформированы требования к разрабатываемой платформе:

- по запросу осуществлять подписку и отписку пользователей от виртуальных сетевых сервисов в рамках модели SaaS;
- при возникновении неисправности принимать меры по восстановлению корректной работы сервиса в автоматическом режиме (healing);
- обеспечивать масштабируемость инфраструктуры сервиса в автоматическом режиме (scaling);
- решение должно быть независимым от платформы виртуализации ресурсов;
- решение должно быть согласовано с высокоуровневой архитектурой ETSI NFV MANO;
- поддерживать работу с несколькими платформами виртуализации ресурсов одновременно.

2. Описание модулей C2 Platform

Рассматриваемая облачная платформа ориентирована на предоставление услуг по модели IaaS. Основной задачей проекта является управление несколькими платформами виртуализации ресурсов (в частности Openstack).

Взаимодействие между внутренними модулями осуществляется через библиотеку RabbitMQ (RMQ) [4].

Проект C2 Platform включает в себя следующие модули:

- GUI-client;
- GUI-server:
- модуль виртуализации инфраструктуры (VIM);
- модуль мониторинга Monitoring (Mon);
- менеджер функций (VNF-M, в разработке);
- менеджер сервисов (VNF-O, в разработке).

Оба модуля (GUI-client и GUI-server) отвечают за отображение актуальной информации о физической и виртуальной инфраструктурах, о состоянии виртуальных сетевых функций и сервисов.

Модуль GUI-server будет использовать интерфейс модуля VNF-O для отображения и управления виртуальными сетевыми функциями и сервисами. Модуль VNF-O взаимодействует напрямую только с GUI-server, поэтому в дальнейшем под GUI будем подразумевать модуль GUI-server.

Модуль VIM состоит из двух основных частей: плагин для существующей платформы виртуализации ресурсов (используется Openstack) и независимая часть для обработки сообщений от других модулей проекта C2 Platform (Mon, VNF-M, GUI-server и т.д.). Плагин можно менять в зависимости от используемой платформы виртуализации ресурсов (Openstack, VMWare и т.д.). На данный момент реализован плагин для работы с Openstack. Задача второй части заключается в управлении тенантами (сеть с виртуальными машинами) по запросу от других модулей платформы.

Модуль мониторинга Mon так же состоит из двух частей: плагин для существующей программы мониторинга (используется Zabbix [5]) и независимая часть для обработки сообщений от других модулей проекта C2 Platform (GUI-server, VNF-M, и т.д.). Плагин можно менять в зависимости от используемой программы мониторинга. Независимая часть обеспечивает слежение за виртуальными машинами, соединениями между ними.

Модуль управления виртуальными сетевыми функциями VNF-M состоит из двух основных частей: анализатор описания виртуальных сетевых функций и часть для взаимодействия с другими модулями проекта C2 Platform (VNF-O, VNF-M). Модуль умеет регистрировать виртуальные сетевые функции, внося соответствующую информацию в базу данных. Модуль анализирует описание функции и занимается конфигурацией ее инфраструктуры.

Модуль оркестратора сетевых сервисов VNF-O отвечает за предоставление точек управления виртуальной инфраструктуры, оркестрацию виртуальных ресурсов и обеспечение жизненного цикла

виртуального сетевого сервиса, обеспечивает его масштабируемость и отказоустойчивость в автоматическом режиме.

В рамках данной работы был разработан модуль оркестратора виртуальных сетевых сервисов для NFV платформы C2 Platform.

3. Пример восстановления сервиса

Рассмотрим пример автоматического восстановления сервиса на примере прокси. Будем использовать виртуальную сетевую функцию прокси-сервер "squid", состоящую из одной виртуальной машины сервиса (на ней и работает ПО прокси "squid"). Рассмотрим следующую ситуацию: клиент NFV-платформы использует виртуальный сетевой сервис прокси. В процессе работы сервиса происходит отказ сервисной виртуальной машины.

На рисунке 2 представлена последовательность основных шагов, которые реализуются в оркестраторе для восстановления сервиса:

- Модуль Мониторинга получает сообщение об отказе виртуальной машине сервиса.
- Мониторинг сообщает в Графический интерфейс о неполадках в виртуальном сетевом сервисе и о начале его восстановления (стрелка с номером 1).
- Мониторинг запрашивает необходимые действия в менеджере функций (стрелка с номером 2).
- Менеджер Функций запрашивает перезагрузку виртуальной машины сервиса в Менеджере Инфраструктуры (стрелка с номером 3).
- Менеджер Инфраструктуры возвращает информацию о перезагруженной машине (стрелка с номером 4).
- Далее Менеджер Функций настраивает перезагруженную машину и сообщает об успешном восстановлении сервиса в Графический Интерфейс (стрелка с номером 5).

Аналогично производится масштабирование виртуального сетевого сервиса.

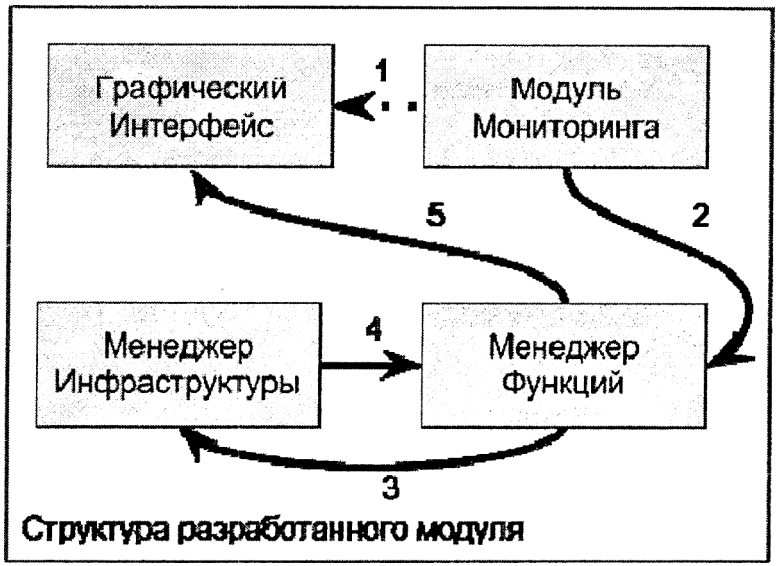

Рис. 2 Пример восстановление сервиса

4. Экспериментальное исследование

Для проверки выполнения всех требований, предъявляемых к NFV платформе было проведено экспериментальное исследование разработанного модуля. Оно заключалось в проведение серии экспериментов, в которых разработанный модуль восстанавливал виртуальные сетевые сервисы по схеме из раздела 4.

Как показало экспериментальное исследование, все требования к разработанному модулю были выполнены и сервис успешно восстанавливался во всех экспериментах.

Заключение

В рамках данной работы был проведен обзор существующих NFV-платформ, на основе которого был сформирован набор необходимых свойств, которым должна удовлетворять NFV платформа. Разработана и реализована архитектура модуля управления виртуальными сетевыми сервисами, обеспечивающего отказоустойчивость и масштабируемость в автоматическом режиме.

Возможные направления дальнейших исследований:

- изменение архитектуры модуля управления виртуальными сетевыми сервисами с целью повышения скорости обработки запросов пользователей на подписку
- разработка классификации событий, связанных с изменением состояния сетевой функции и сервиса

Литература

1. Xin M., Levina N. Software-as-a-service model: Elaborating client-side adoption factors //Proceedings of the 29th International Conference on Information Systems, R. Boland, M. Limayem, B. Pentland,(eds), Paris, France. – 2008.

2. ETSI Industry Specification Group, ETSI GS NFV-IFA 001 V1.1.1, 12.2014

3. ETSI Industry Specification Group, ETSI GS NFV-IFA 010 V2.1.1, 04.2016

4. Boschi S., Santomaggio G. RabbitMQ Cookbook. – Packt Publishing Ltd, 2013.

5. Olups R. Zabbix 1.8 network monitoring. – PACKT Publishing Ltd, 2010.

Раздел III.
Системы реального времени

Балашов В.В., Костенко В.А.

Комплексы бортового радиоэлектронного оборудования с архитектурой ИМА[1]

В настоящее время многие комплексы бортового радиоэлектронного оборудования (БРЭО) строятся в соответствии с федеративной архитектурой или интегрированной модульной архитектурой [1]. В комплексах БРЭО выделяют три уровня обработки данных [2]:

1) уровень предобработки входных данных,
2) уровень первичной обработки данных,
3) уровень вторичной обработки данных.

Для обеспечения выполнения функциональных задач (программ) в режиме реального времени (выполнение с требуемой частотой или в рамках заданного директивного интервала) наибольшая производительность необходима на уровне первичной обработки данных. Программы первичной обработки данных каждой системы в комплексах БРЭО с федеративной архитектурой выполняются на своих вычислительных модулях. Для обеспечения требуемой производительности очень часто используются специализированные вычислительные модули. На этих вычислительных модулях не могут выполняться программы других систем.

Наиболее широко используемый подход к построению комплексов БРЭО с интегрированной модульной архитектурой известен как интегрированная модульная авионика (ИМА) [3]. Основной особенностью ИМА является унификация аппаратных и программных средств используемых в комплексах БРЭО. Программы первичной обработки данных различных систем в комплексах с архитектурой ИМА могут разделять вычислительные модули. При таком подходе к построению комплексов БРЭО требуется обеспечить изоляцию программ различных систем. Изоляция должна распространяться на все ресурсы, включая регистровую память, кэши центральных процессоров, каналы ввода-вывода.

Для поддержки требований реального времени и изолированности программ различных систем друг от друга был разработан стандарт ARINC 653 [4]. Изолированность программ

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

различных систем обеспечивается введением разделов и окон. Для программ каждой системы выделяется свой раздел и набор временных окон (непересекающихся интервалов времени). Программы раздела могут выполняться только в рамках своих окон и каждому разделу выделяется необходимая память, к которой не могут обращаться программы других разделов. Программы раздела внутри окна запускаются на выполнение по мере готовности данных в соответствии с приоритетами. Допустимо прерывание программы и ее последующее выполнение в этом окне или в одном из следующих окон раздела. Программы различных разделов могут взаимодействовать только путем передачи сообщений. Требования работы программ в режиме реального времени обеспечиваются построением расписания закрытия и открытия окон для каждого ядра вычислительных модулей.

Российской операционной системой соответствующей стандарту ARINC 653 является ОС РВ Багет 3.0 [5, 6]. В ОС РВ Багет 3.0 стандарт ARINC 653 выбран в качестве основного. Реализованы все обязательные функции ARINC 653. Стандарт POSIX используется в той мере, в какой это не противоречит ARINC 653.

Бортовая сеть обмена данными комплексов БРЭО с архитектурой ИМА летательных аппаратов гражданской авиации строится на основе коммутируемой сети AFDX, военной авиации – на основе базовых топологий FC [7]: точка-точка, коммутируемая сеть, кольцо с арбитражем. Построение этих сетей регламентируют соответственно стандарты ARINC 664 (AFDX) и FC-AE-ASM-RT. Основные принципы построения бортовых коммутируемых сетей обмена рассмотрим на примере AFDX.

Стандарт Avionics Full Duplex Ethernet (AFDX) [8] описывает управление бортовыми сетями на основе традиционного стандарта Ethernet 802.3. Согласно стандарту AFDX, сеть состоит из следующих элементов: абоненты, передающие сообщения; оконечные системы – системы, обеспечивающие интерфейс между абонентами и сетью; пакетные коммутаторы, соединяемые линиями передачи данных.

Каждый абонент подключен к оконечной системе, причем к одной оконечной системе может быть подключено несколько абонентов. Соблюдение ограничений на время передачи сообщений в сети AFDX достигается за счет выделения гарантированной пропускной способности соединению между каждой парой оконечных систем. Такое соединение может проходить через несколько пакетных коммутаторов и линий передачи данных. В AFDX соединение между оконечными системами называют виртуальным каналом. Передача данных между абонентами осуществляется путем передачи сообщений по виртуальным каналам, маршруты которых в физической сети определены заранее. Для каждого виртуального канала определена только одна оконечная система-отправитель и одна или более оконечная

система-получатель. При этом по одному виртуальному каналу могут передаваться сообщения только от одного абонента.

Сообщение от абонента попадает на оконечную систему через специально выделенный порт. На оконечной системе-отправителе сообщение разбивается на порции данных (кадры), которым сопоставляются заголовки соответствующих уровней стека TCP/IP – а именно, транспортного (UDP), сетевого (IP), канального. Полученные в результате разбиения кадры выдаются в физический канал передачи данных. В каждый кадр, в поле MAC-адреса получателя записывается номер виртуального канала. Этот номер используется коммутаторами AFDX для маршрутизации кадра. Стандартная для Ethernet маршрутизация по MAC-адресу в AFDX не используется. После доставки кадра на оконечную систему-получатель (возможно, одну из нескольких) он буферизуется для последующей сборки сообщения. После прихода последнего кадра собранное сообщение направляется абонентам-получателям. Маршруты виртуальных каналов в сети AFDX задаются статически, возможность динамически изменять таблицы маршрутизации стандартом не предусмотрена. Для контроля пропускной способности виртуального канала используют алгоритм текущего ведра с маркерами [9].

Основные причины, которые могут приводить к снижению эффективности использования аппаратных средств вычислительных и сетевых ресурсов комплексов БРЭО с архитектурой ИМА следующие:

1) использование универсальных вычислительных модулей разделяемых программами первичной обработки информации различных систем комплекса,

2) использование только языков программирования высокого уровня для достижения унификации программных средств,

3) увеличение потока данных в бортовой сети обмена при переносе программ первичной обработки данных с вычислителя системы на универсальные вычислительные модули.

Также следует отметить, что значительно возрастает сложность этапа комплексирования систем. На этапе комплексирования систем ИМА возникают, в частности, следующие задачи:

1) распределение вычислительной нагрузки по модулям и процессорным ядрам системы с минимизацией загрузки бортовой сети,

2) конфигурирование бортовой сети, включая формирование системы виртуальных каналов и расчет их характеристик, построение маршрутов виртуальных каналов в бортовой сети обмена;

3) построения расписаний выполнения вычислений на процессорах в составе системы ИМА.

Для решения перечисленных задач, в Лаборатории вычислительных комплексов факультета ВМК МГУ разработаны инструментальные системы [10-12]:

- САПР функциональных задач.
- САПР AFDX.
- САПР циклограмм.

Ограничения на корректность решений в САПР функциональных задач определяются требования стандарта и особенностями российской операционной системы реального времени (ОС РВ) Багет 3.0 соответствующей стандарту ARINC 653.

Ограничения на корректность решений в САПР AFDX определяются требованиями стандарта Avionics Full Duplex Ethernet (AFDX) [8].

Расписания, построенные САПР циклограмм, совместимы с адаптерами МКИО / MIL STD-1553B, поддерживающими выполнение цепочек работ. Такие адаптеры поставляются компаниями Элкус, DDC, Condor Engineering, и предназначены для функционирования под управлением ОС РВ «Багет», QNX, VxWorks, а также ОС Linux с расширениями реального времени. САПР циклограмм применяется для поддержки комплексирования БРЭО современных летательных аппаратов, а также морских навигационных комплексов. САПР интегрирована в общий технологический процесс проектирования комплексов БРЭО.

Использование подобных САПР необходимо для сокращения сроков и стоимости проектирования комплексов БРЭО с архитектурой ИМА. Также целесообразна разработка высокоуровневой САПР систем ИМА, отвечающей за согласованное решение задач 1 – 3 и определение программ первичной обработки данных, которые следует переносить с вычислителя системы на универсальные вычислительные модули.

Литература

1. Парамонов П. П., Жаринов И.О. Интегрированные бортовые вычислительные системы: обзор современного состояния и анализ перспектив развития в авиационном приборостроении // Научно-технический вестник информационных технологий, механики и оптики, 2013, № 2 (84), С. 1-17.

2. Тропченко А.Ю, Тропченко А.А. Цифровая обработка сигналов. Методы предварительной обработки. Учебное пособие по дисциплине «Теоретическая информатика» // СПб: СПбГУ ИТМО, 2009. – 100 с.

3. ARINC 651-1 “Design Guidance for Integrated Modular Avionics”, 1997.

4. Arinc Specification 653. Airlines Electronic Engineering Committee. [PDF] (http://www.arinc.com).

5. Годунов А.Н., Операционные системы реального времени Багет 3.0. // Программные продукты и системы, 2010., № 4. С. 15-19.

6. Годунов А.Н., Солдатов В.А., Операционные системы семейства Багет (сходство, отличия и перспективы) // Программирование, 2014., № 5. С. 69-76.

7. INCITS 373. Information Technology -Fibre Channel Framing and Signaling Interface (FC-FS), International Committee for Information Technology Standards, 2003.

8. Aircraft Data Network. Part 7. Avionics Full Duplex Switched Ethernet (AFDX) Network. // Aeronautical Radio, Inc. – 2012.

9. Смелянский Р. Л.. Компьютерные сети (т. 2). М.: Академия, 2011. - 240 с.

10. 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.

11. Вдовин П.М., Костенко В.А. Исследование эффективности процедуры агрегации виртуальных каналов при построении бортовых коммутируемых сетей // Вестн. Моск. ун-та. Сер. 15. Вычислительная математика и кибернетика. - 2015. - №. 4 - С. 32-40.

12. Р. Смелянский, В. Костенко, В. Балашов, В. Балаханов. Инструментальная система построения расписания обмена данными по каналу с централизованным управлением // Современные технологии автоматизации. - 2011. - № 3 - С.78-84.

Шпилевой В.Д., Герасёв А.В.

Исследование и модификация алгоритма синхронизации времени от источника сигнала эталонной частоты в ядре Linux[1]

1. Постановка задачи

В настоящее время в различных областях активно используются вычислительные системы, работающие в режиме реального времени. Это может быть мобильный телефон, контроллер стиральной машины или управляющий компьютер автомобиля. Такие системы со временем становятся сложнее, в их работе начинают использоваться несколько вычислителей объединенных каналами связи, так что они превращаются в распределенные вычислительные системы реального времени (РВС РВ).

Примером такой системы является стенд комплексирования и приемосдаточных испытаний (СК ПСИ) информационно-управляющей системы (ИУС) самолета. Данный стенд являтся программно-аппаратным комплексом, реализующим полунатурное имитационное моделирование окружения ИУС для решения задач интеграции и тестирования аппаратуры и ПО ИУС. [2]

С одной стороны, количество бортовых каналов современной ИУС исчисляется сотнями, а количество различных сигналов принимаемых и передаваемых ИУС — сотнями тысяч. Таким образом для функционирования среды моделирования необходимо достаточно большое число вычислителей.

С другой стороны, частоты данных каналов достигают гигагерц, а для успешной имитации циклограмм бортового обмена требуется высокая точность синхронизации выдачи данных по различным интерфейсам.

Для этого, в свою очередь, требуется синхронизация часов вычислителей между собой с точностью сотен или десятков микросекунд.

Подробнее понятие точности синхронизации рассмотрено в разделе 2.

Программная платформа СК ПСИ построена на основе ОС Linux с модификациями реального времени, поэтому для синхронизации часов системы может быть использована реализация

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

NTP (Network Time Protocol) [4], но таким образом можно получить точность синхронизации до единиц миллисекунд [7], чего, как было показано выше, недостаточно.

Для достижения более высокой точности может быть использована синхронизация времени по источнику сигнала эталонной частоты [1]. Источник раз в секунду с высокой точностью генерирует сигнал (pulse per second — PPS), все компоненты системы подключены к этому источнику по интерфейсу, минимизирующему задержки получения сигнала (GPIO, RS-232, IEEE 1284 и т. п.), и осуществляют синхронизацию своих локальных часов по этому сигналу. Таким источником может являться приемник спутникового сигнала (GPS), специальный прибор с кварцем высокой точности или даже один из вычислителей системы, так как для системы может быть важна не абсолютная точной хода часов на разных ее узлах, а только их относительная синхронизация.

В ядре ОС Linux существует реализация механизма синхронизации времени по сигналу эталонной частоты — подсистема HardPPS [1] и набор драйверов для подключения устройств-источников PPS. Данная реализация даёт высокую (порядка микросекунды) точность синхронизации, однако было замечено, что в ряде случаев используемый подсистемой алгоритм начинает терять точность и расхождение часов на узлах РСВ достигает десятков и даже сотен миллисекунд. Так как такое поведение алгоритма возникает в том числе на входных данных, встречающихся в реальной жизни, возникла необходимость исследовать поведение этого алгоритма, выяснить какие аномалии входных данных приводят к снижению точности синхронизации и предложить модификации алгоритма, устойчивые к таким входным данным. Данная статья посвящена решению этих задач.

2. Задача синхронизации времени

Задача синхронизации времени состоит в применении различных механизмов, обеспечивающих такой ход локальных часов узлов РВС, что любой момент времени разница показаний часов на любых двух узлах не превышала некоторую величину t, называемую точностью синхронизации. Чем ниже можно выбрать эту величину, тем соответственно, точнее (с более высокой точностью) синхронизированы часы на узлах РВС.

Чтобы приступить к обзору алгоритма синхронизации, реализованного в HardPPS, необходимо разобраться, как устроена работа со временем в Linux, а так же как возникают и обрабатываются PPS-сигналы.

2.1. Время в Linux

На физическом уровне время в компьютере представлено, как значение некоторого постоянно возрастающего аппаратного счетчика, значение которого увеличивается через примерно равные промежутки времени. Счетчик не может инкрементироваться через точно равные промежутки времени по аппаратным причинам [3].

Значение этого счетчика в каждый момент времени - целое число, которое используется системой для перевода его в астрономические единицы измерения времени.

Значение счетчика - это «необработанное» значение времени. Так же в системе есть понятие «реального времени» - это время, получаемое из необработанного с помощью различных коэффициентов пересчета. Именно это время используется прикладным и системным ПО в его работе и называется «системным».

Применение алгоритмов синхронизации времени - это способ улучшить коэффициенты пересчета таким образом, чтобы в любой момент времени реальное время, полученное из необработанного с использованием этих коэффициентов, было как можно ближе к эталону.

2.2. Pulse per second

При получении сигнала PPS аппаратура формирует прерывание, драйвер PPS-источника засекает текущее «необработанное» время и передает эту информацию в подсистему HardPPS, где она используется для модификации коэффициентов пересчета.

Так как на всех этапах (обработка сигнала, формирование и доставка прерывания, передача управления обработчику прерывания и т. п.) возможно различные недетерминированные задержки, то применяются различные меры для минимизации этих задержек и более точной засечки момента времени. [11][12]Однако даже использование этих мер не избавляет от спорадических выбросов задержки, что «загрязняет» эти данные достаточно сильным шумом.

Подсистема HardPPS на основании меток времени возникновения сигнала PPS вычисляет коэффициенты пересчета. Для этого используется подход основанный на алгоритме NTP.

2.3. Алгоритм коррекции времени NTP

Одним из наиболее часто применяемых на практике механизмов синхронизации времени является протокол NTP, описанных в стандарте RFC 5905. Данный стандарт регламентирует как сетевое взаимодействие, так и непосредственно описывает

алгоритм, применяя который можно, модифицируя коэффициенты пересчета, «замедлять» или «ускорять» ход системных часов, не допуская для «системного» времени обратного хода, резких скачков и т. п. Подробное описание алгоритма приведено непосредственно в стандарте .[4].

Рассмотрим задачи, которые решает алгоритм NTP. Это устранение расхождения частоты хода часов (относительно эталона) и фазы часов.

Под частотой часов системы понимается величина, на которую изменяется значение аппаратного счетчика за одну секунду. Так как эта величина используется для вычисления «системного» времени, то ее необходимо откалибровать по эталону, например по сигналам PPS-источника. При этом отклонение частоты от эталонной называется ошибкой частоты хода часов.

Исправление ошибки частоты позволяет добиться синхронной с эталоном скорости хода системных часов и избежать расхождение часов в дальнейшем, но при этом между эталонным временем и системным останется некоторая константная разница. Эта величина называется ошибкой фазы хода часов.

Так как в силу физической природы счетчика времени существует флуктуация его собственной частоты, то процесс устранения расхождения частоты и фазы часов должен выполняться постоянно во время работы системы.

Для корректировки частоты все время работы системы разбивается на интервалы, которые называются интервалами частоты. По величине отклонения времени от эталона, измеренной в начале и конце данного интервала, выполняется вычисление значения коррекции частоты (по сути линейного коэффициента — скорости расхождения часов), которое затем применяется весь следующий интервал. Так как удлинение интервала позволяет уменьшить влияние ошибки измерения, лучше начинать работу алгоритма с короткого интервала для быстрого получения грубой оценки ошибки частоты и увеличивать его в дальнейшем. С другой стороны, в случае изменения частоты часов системы (например из-за изменения частоты генератора (кварца), что часто случается при увеличении вычислительной нагрузки и повышении температуры системы) стоит уменьшить интервал частоты, чтобы его начало и конец приходились на одну и ту же, актуальную частоту.

В отличие от частоты, коррекгировка фазы выполняется каждую секунду, так как неисправленные ошибки фазы суммируются, накладываясь друг на друга, и их необходимо вовремя исправлять. При этом для обеспечения плавности этого исправления, корректировка выполняется равномерно за несколько тактов работы в течении этой секунды.

Для вычисления величины корректировки используются значения ошибки фазы за последние несколько секунд и, для устранения ошибки измерения, некоторый алгоритм фильтрации, выбор которого зависит от реализации.

3. Подходы к анализу алгоритма синхронизации времени

Прежде чем переходить к исследованию работы алгоритма синхронизации времени и его модификациям, необходимо разработать методику анализа работы этого алгоритма. Первичной численной величиной для такого анализа является величина расхождения эталонного времени и системного, которая может быть достаточно точно измерена в момент прихода PPS-сигнала. Эта величина является входными данными для алгоритма.

Стоит отметить, что работа подсистемы синхронизации времени зависит не только от последовательности входных данных, но и от ряда внутренних факторов, таких как текущая частота системного счетчика и его значения в моменты времени, когда возникают внешние для системы события (сигналы PPS).

Таким образом для анализа работы алгоритма необходим профилировщик работы подсистемы, работающий в пространстве ядра и собирающий трассу входных данных, внутренних параметров подсистемы и последовательности выполняемых шагов. При этом реализация профилировщика не должна влиять на временные характеристики работы ядра. Такой профилировщик для подсистемы HardPPS ядра Linux был создан [11] в рамках данной работы и использовался при исследовании и сравнении алгоритмов.

4. Анализ работы алгоритма синхронизации

В первую очередь в рамках исследования был проведен анализ трасс работы существующего алгоритма в случае возникновения существенного нарушения точности синхронизации. Во всех случаях причиной ухудшения точности являлось неправильное вычисление ошибки частоты.

В большинстве случаев причиной было резкое изменение разницы частот источника и потребителя сигналов в результате изменения температуры потребителя или изменения частоты источника. Рассмотрим природу происходящего.

Если частота будет монотонно изменяться небольшими шагами каждые 2-3 секунды, например, в сторону убывания, то алгоритм считает такую частоту стабильной и продолжает увеличивать длины своих интервалов частоты. Такое поведение приводит к тому, что уже спустя 10-15 секунд после начала очередного длинного

интервала частоты точность синхронизации падает с порядка наносекунд, до десятков, сотен микросекунд, или даже до миллисекунд, а до ближайшей коррекции частоты может пройти несколько минут. И даже после ее очередной коррекции длина интервала либо снова увеличивается, либо уменьшается недостаточно сильно, и точность синхронизации продолжает ухудшаться.

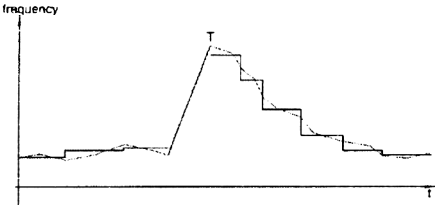

Рис. 1. Пример монотонного убывания частоты

На рисунке 1 изображен пример того, как стабильность частоты может недостаточно адекватно оцениваться алгоритмом ее коррекции. Здесь изображена работа алгоритма на некоторых данных. На ось ординат отображаются значения частоты, высчитанной на каждом интервале частоты, а на ось абсцисс - время работы алгоритма. Серым обозначена реальная частота, а синим - какой частоту считает алгоритм корректировки частоты. До момента времени T эталонный источник частоты работал относительно стабильно, и алгоритм обновлял ее с приемлемой точностью. Но пусть в момент времени T эталонный источник был, например, перезагружен, или перегрелся, в результате чего частота резко увеличилась и сразу начала медленно спадать обратно.

В момент T интервал частоты сбрасывается, согласно алгоритму, а затем работа возобновляется. Пусть частота теперь монотонно уменьшается так, что, это не приведет к уменьшению длин интервалов частот, в результате чего просчитанная и используемая в алгоритме для корректировки частота может сильно отличаться от реальной, и будет обновляться слишком редко.

Так же уже в процессе аналитического анализа алгоритма была выявлена еще одна проблема, которую не удалось получить экспериментально, но это не означает ее отсутствие:

Если текущий интервал частоты достаточно длинный (в существующей реализации он может достигать 256 сек.) и закончится относительно нескоро, а диапазон, в котором колеблется частота, вдруг немного сдвинется в какую-либо сторону (см. Рис.2), то точность синхронизации ухудшится как минимум до конца такого интервала.

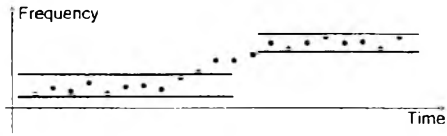

Рис. 2. Пример сдвига диапазона колебания частоты

5. Модификации алгоритма

5.1. Фильтр Калмана

В качестве первой модификации был опробован Фильтр Калмана - одна из широко распространенных техник оценки состояния динамической системы по неполным или зашумленным данным [**Ошибка! Источник ссылки не найден.**]. Фильтр чрезвычайно прост в реализации и показывает при этом хорошие результаты во многих областях применения. Однако попытка применения фильтра Калмана оказалась неудачной, так как с его использованием не удалось добиться баланса между хорошей фильтрацией «шума» (выбросов, в измеряемых данных, связанных в первую очередь с погрешностью измерений) и низкой реактивностью системы. В тоже время модификации, описанные ниже, показали значительно лучшие результаты.

5.2. Анализ последних значений частоты

В рамках данного подхода при каждом возникновении PPS-сигнала предлагается запоминать не только очередную ошибку фазы, но и текущую ошибку частоты. То есть алгоритм должен хранить вычисленные отклонения частоты от эталонной за последние несколько секунд.

Известно, что алгоритм в каждый момент времени находится в некотором интервале частоты, а также известно, какую коррекцию частоты на этом интервале применяет алгоритм - она была посчитана в конце предыдущего интервала. Значит, в каждый момент времени можно узнать, насколько коррекция частоты текущего интервала далека от ошибок частоты за последние несколько секунд. Так можно определять, насколько адекватна текущая коррекция ошибки.

Алгоритм с данной модификацией принимает за меру стабильности частоты минимальную по модулю разность текущей коррекции частоты и последних ошибок частоты. Если такая разность оказывается больше некоторой величины, то значит частота изменилась настолько, что применяемая коррекция уже неверно исправляет ее. В таком случае текущий интервал частоты прерывается, но его результаты не отбрасываются, как в случае оригинального алгоритма, а продолжают использоваться до вычисления следующего

значения. При чем длина следующего интервала частоты принимается в два раза меньше длины текущего.

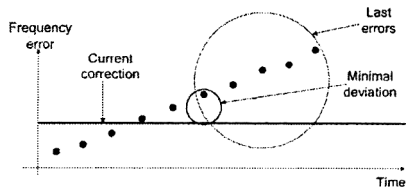

Рис. 3. Пример работы модификации на нестабильной частоте

На рисунке 3 изображен пример того, как алгоритм модификации может реагировать на монотонное изменение частоты. Здесь так же хранятся последние 5 ошибок частоты, и на каждом шаге считается расстояние от текущей коррекции до ближайшей ошибки. Но в данном примере это расстояние уже достаточно велико, из чего можно заключить, что коррекция уже плохо обновляет частоту.

5.3. Инверсии ошибок фазы

Инверсия последовательности чисел - это такая пара чисел x и y в ней, что x расположен в ней раньше, чем y, но при этом $x > y$ [5]. Инверсии же ошибок фазы определим по аналогии, но с поправкой - не $x > y$, а $|x| > |y|$. То есть будем сравнивать по абсолютным значениям.

При таком определении инверсии ошибок фазы большое количество инверсий в последовательности ошибок фазы свидетельствует о том, что ошибки фазы уменьшаются, то есть алгоритм синхронизации времени сходится. Если же инверсий мало, то ошибки фазы возрастают по модулю, значит качество синхронизации с эталонным источником ухудшается.

Алгоритм с данной модификацией проверяет в момент возникновения PPS-сигнала количество инверсий в последовательности из последних ошибок фазы. Если количество инверсий меньше некоторого порога, то это означает, что величина ошибок фазы в целом возрастает, а значит коррекция фазы не поспевает за скоростью изменения счетчика времени. В таком случае предлагается скорректировать частоту тем же способом, что в предыдущей рассмотренной модификации.

6. Экспериментальная апробация

Для сравнения был выбран ряд характеристик:

- время сходимости алгоритма
- математическое ожидание ошибки фазы
- среднеквадратичное отклонение ошибки фазы

Для анализа выбраны именно ошибки фазы, так как ошибки частоты так же на них отражаются, но не наоборот.

В качестве данных для анализа были использованы записи реальной работы алгоритма на стабильной частоте, а также на аномальных входных данных.

Временем сходимости здесь называется время с начала работы алгоритма, спустя которое первый раз на трех PPS-сигналах подряд смещение фазы было менее 2000 наносекунд. Такая граница была выбрана после анализа нормальной работы алгоритма на стабильной частоте, так как в этом случае почте все ошибки фазы по модулю не превышают это значение.

6.1. Стабильная частота

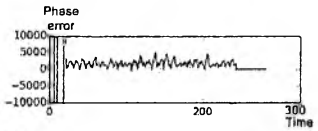

Рис. 4. Применение анализа последних частот

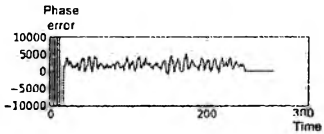

Рис. 5. Применение анализа инверсий последних ошибок фазы

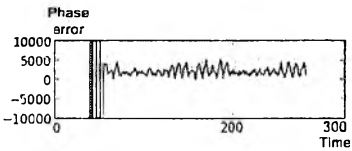

Рис. 6. Оригинальный алгоритм без модификаций

Как видно из графиков (Рис. 4-6), на стабильной частоте модификации показали те же результаты, что и оригинальный алгоритм.

Таблица 1

Измерения эффективности на стабильной частоте

Тип алгоритма	Длина записи	Математическое ожидание	Среднее отклонение	Время сходимости
Анализ последних частот	244 сек.	1605 нс.	268 нс.	22 сек.
Анализ количества инверсий ошибок фазы	248 сек.	1756 нс.	335 нс.	18 сек.
Оригинальный алгоритм	280 сек.	1746 нс.	263 нс.	60

6.2. Нестабильная частота

Исследование не нестабильной частоте проводилось на системе из двух машин и одного PPS-источника, к которому они были подключены. На каждой машине работала синхронизация времени с PPS-источником, но на одном работала реализация оригинального алгоритма, а на другом реализация одной из модификаций.

6.2.1. Анализ последних значений частоты

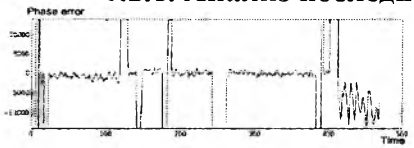

Рис. 7. Оригинальный алгоритм без модификаций

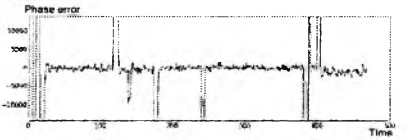

Рис. 8. Применение анализа последних частот

Из графиков можно заметить, что оригинальный алгоритм после 400-й секунды стал сильно расходится, в то время как модифицированный лишь едва заметно поколебался. Измерения характеристик эффективности представлены в таблице 2.

Таблица 2

Тип алгоритма	Длина записи	Математическое ожидание	Среднее отклонение	Время сходимости
Оригинальный алгоритм	470 сек.	-1359 нс.	2989 нс.	15 сек.
Анализ последних частот	468 сек.	-160 нс.	688 нс.	23 сек.

Из таблицы можно сделать вывод, что модификация анализа последних частот улучшила качество синхронизации на порядок.

6.2.2. Инверсии ошибок фазы

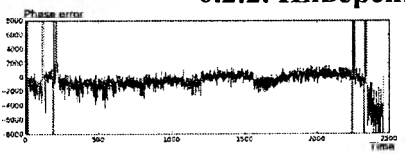

Рис. 9. Оригинальный алгоритм без модификаций

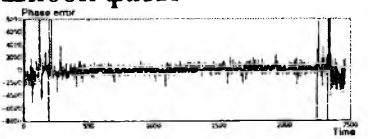

Рис. 10. Применение анализа инверсий последних ошибок фазы

Сравнивая графики на рисунках 9 и 10 можно заметить, что модифицированный алгоритм не только ведет себя более стабильно под конец работы, но и в целом по всему времени его работы он

меньше "дрожит". Измерения характеристик эффективности представлены в таблице 3.

Таблица 3

Тип алгоритма	Длина записи	Математическое ожидание	Среднее отклонение	Время сходимости
Оригинальный алгоритм	2126 сек.	-524 нс.	694 нс.	22 сек.
Анализ инверсий ошибок фазы	2171 сек.	-44 нс.	498 нс.	17 сек.

Аналогично модификации анализа последних частот, анализ инверсий ошибок фазы так же показал улучшение точности синхронизации на порядок.

7. Результаты

В работе исследована работа существующей в ядре ОС Linux подсистемы синхронизации времени HardPPS на реальных вродных данных. Предложены характеристики качества для анализа и сравнения алгоритмов и разработан инструментарий для более детального исследования работы алгоритмов в рамках этой подсистемы.

В результате данного исследования выявлены причины, приходящие в ухудшению точности синхронизации и предложен ряд модификаций алгоритма, призванных нивелировать влияния этих причин.

Данные модификации были реализованы и проведена их апробация на реальных входных данных, которая показало для двух из них существенное (на порядок) улучшение точности синхронизации на аномальных входных данных при сохранении точности и скорости сходимости в нормальной ситуации.

Литература

1. Mogul J. C. et al. Pulse-per-second api for unix-like operating systems, version 1.0 //Pulse. – 2000,

2. A hardware-in-the-loop simulation environment for real-time systems development and architecture evaluation / V. V. Balashov, A. G. Bakhmurov, M. V. Chistolinov et al. // Proc. 3rd International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX'2008). — 2008. — P. 80–86,

3. Gleixner T., Niehaus D. Hrtimers and beyond: Transforming the linux time subsystems //Proceedings of the Linux symposium. – 2006. – T. 1. – C. 333-346,

4. Mills D. L. Internet time synchronization: the network time protocol //Communications, IEEE Transactions on. – 1991. – T. 39. – №. 10. – C. 1482-1493,

5. Pemmaraju S., Skiena S. S. Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica – Cambridge university press, 2003.

6. Welch G., Bishop G. An introduction to the kalman filter. Department of Computer Science, University of North Carolina. – 2006.

7. Mills D. L. On the accuracy and stablility of clocks synchronized by the network time protocol in the Internet system //ACM SIGCOMM Computer Communication Review. – 1989. – T. 20. – №. 1. – C. 65-75.

8. Mumford P. J. Relative timing characteristics of the one pulse per second (1PPS) output pulse of three GPS receivers //Proc. 6th International Symposium on Satellite Navigation Technology Including Mobile Positioning & Location Services. – 2003. – C. 22-25

9. Siccardi M., Abgrall M., Rovera G. D. About time measurements //2012 European Frequency and Time Forum. – 2012

10. Mills D. A kernel model for precision timekeeping. – 1994

11. В. Д. Шпилевой Исследование и модификация алгоритма синхронизации времени от источника сигнала эталонной частоты в ядре Linux. Выпускная квалификационная работа. //Московский государственный университет имени М.В. Ломоносова, 2016. [PDF] (https://cloud.lvk.cs.msu.su/index.php/s/liyVPagVTPkAdkB)

12. В. Д. Шпилевой Исследование механизмов синхронизации времени от различных аппаратных источников в ядре Linux. Курсовая работа //Московский государственный университет имени М.В. Ломоносова, 2015. [PDF] (https://cloud.lvk.cs.msu.su/index.php/s/dkER6OU4gTB6hjq)

Новоселов А.Д., Балаханов В.А.

Построение статико-динамического расписания алгоритмом с обратной связью[1]

Введение

Задача построения статико-динамического расписания возникает при разработке вычислительных систем реального времени, построенных в соответствии с концепцией интегрированной модульной авионики (ИМА). Для операционных систем, работающих по концепции ИМА, был разработан стандарт ARINC 653[2], специфицирующий прикладной интерфейс операционной системы.

На настоящий момент в рамках Лаборатории Вычислительных Комплексов решена задача построения статико-динамического расписания [3] и существует программная реализация алгоритма составления расписания. Помимо алгоритма составления расписания имеется модель вычислений, необходимая для валидации корректности расписания, т.е. проверки наличия выполнения всех работ из исходного набора в рамках своих директивных интервалов.

Необходимость в модели вычислений обусловлена динамической составляющей процедуры планирования в системе: работы в рамках разделов ставятся на выполнение динамически, а само расписание не содержит и не должно содержать информацию о точном времени запусков работ.

Система Интегрированной Модульной Авионики (ИМА) - это распределенная система реального времени, состоящая из вычислительных модулей, соединенных средой передачи данных. Типичная система ИМА использует несколько типов унифицированных модулей и коммутируемую сеть с поддержкой виртуальных каналов для передачи данных. Каждый модуль содержит один или несколько многоядерных процессоров.

Единицей планирования ресурсов для систем ИМА является раздел. Понятие раздела является ключевым в стандарте ARINC653 [2]. Раздел - это, по сути, приложение, выполняющее определенную прикладную задачу. Рассматриваемая архитектура, лежащая в основе разбиения на разделы в системе интегрированной модульной авионики, характеризуется следующими свойствами: каждый раздел состоит из одного или нескольких одновременно выполняемых процессов, совместно использующих ресурсы процессора; процессорные ресурсы

[1] Работа выполнена при финансовой поддержке РФФИ, грант N 16-07-01237.

выделяются статически для каждого раздела в форме временных интервалов - окон.

Окном является временной интервал, в течение которого задачи конкретного раздела могут быть выполнены на ядре процессора, к которому раздел привязан. Окна, привязанные к одному модулю, не пересекаются во времени.

Сетка исполнения окон одинакова для всех ядер одного и того же модуля. Для обеспечения надёжности и безопасности системы все взаимодействия между разделами сводятся к обмену сообщениями. Взаимное влияние разделов друг на друга сводится к минимуму.

Как было показано в работе [3], задача построения статико-динамического расписания является NP-трудной и поэтому не известен алгоритм, которой бы являлся оптимальным. Существующий алгоритм построения расписания состоит из нескольких этапов:

1. объединение задач в разделы;
2. привязка разделов к ядрам;
3. построение набора окон для модулей;
4. определение приоритетов задач в рамках раздела;
5. проверка корректности расписания при помощи модели вычислительной системы.

Представляет интерес именно второй этап алгоритма, а именно привязка разделов к ядрам. Дело в том, что в работе [3] используется жадный алгоритм, согласно целевой функции которого, необходимо было минимизировать трафик сообщений между зависимыми по данным задачами. Проблема заключается в том, что поэтапное построение расписания не позволяет учесть на этапе привязки разделов к ядрам все ограничения. Таким образом, существует вероятность, что невозможность построения корректного расписания может быть заложена на этапе привязки разделов к ядрам и дальнейшие шаги алгоритма не приведут к построению корректного расписания. Зато может представлять интерес результат работы модели вычислений, которая содержит информацию о невыполненных работах.

Автором данной статьи выдвинуто предположение о том, что, проанализировав результат работы модели, можно определить причины того, что работы не успевают выполниться. Тогда, используя эту информацию, можно перераспределять разделы по ядрам, чтобы минимизировать количество не успевших вовремя выполниться работ и в идеале добиться выполнения всех работ. На основе данного предположения схема работы алгоритма может быть дополнена введением обратной связи – результаты работы поздних этапов

построения расписания будут использоваться для корректировки работы ранних этапов на следующей итерации:

6. анализ результатов работы модели и выявление причин того, что работы не успевают выполниться;
7. переход к п.2 (привязка с учетом результатов анализа).

В данной статье рассматривается алгоритм построения статико-динамического расписания с обратной связью, основанный на приведенной выше схеме.

Постановка задачи

Расписание для системы ИМА определяет набор окон исполнения для каждого модуля и привязку разделов к окнам для каждого ядра. Расписание строится для длительности интервала планирования, которое соответствует наименьшему общему кратному периодов задач. Расписание строится статически и загружается в систему ИМА как часть ее конфигурации.

Поскольку задачи имеют, как правило, разные периоды, алгоритм построения расписания оперирует таким понятием, как работа (экземпляр задачи). Для каждой периодической задачи существует множество работ. Директивный интервал работы определяется порядковым номером итерации выполнения работы и периодом задачи, соответствующей работе.

Работы принимают входные данные и генерируют выходные данные в виде сообщений, так что множество периодических задач дополняется набором сообщений.

Если задачи отправителя и получателя сообщения принадлежат разным разделам, размещенным на одном и том же модуле, сообщения между этими задачами передаются через память модуля. Передача сообщения через память быстрее, чем передача по каналу.

Работа является неготовой к выполнению в случае, если прибыли не все сообщения от тех работ, от которых она зависит по данным. После того, как выполнение задачи завершено, начинается передача всех своих сообщений. Если работа начинается в пределах некоторого окна, но не закончена до конца окна, ее выполнение возобновляется в следующем окне одного и того же раздела, опять же в соответствии с приоритетом задачи.

В системе ИМА после окончания интервала планирования, расписание выполнения окон выполняется снова. При этом в расписании не содержится никакой информации о том, в какой последовательности будут выполняться работы в одном окне, а в окнах

работы ставятся на выполнение динамически в соответствии с заданными приоритетами.

Расписание не содержит информации о конкретных интервалах выполнения работ. Поэтому для определения корректности расписания используется вычислительная модель системы, позволяющая построить временную диаграмму выполнения расписания. Для разработки алгоритма с обратной связью для решения задачи построения статико-динамического расписания необходимо в первую очередь проанализировать возможные причины невыполнения работ в рамках своих директивных интервалов и предложить алгоритм корректировки расписания. Требования, предъявляемые алгоритму: алгоритм должен определять причины невыполнения работ на основе анализа временной диаграммы, построенной моделью вычислительной системы, и на основе результатов анализа вносить корректировки в привязку разделов к ядрам. В результате работы алгоритма должно получиться расписание, в котором выполнятся все запланированные работы.

Входные данные.

Для описания входных данных задачи построения статико-динамического расписания мы воспользуемся обозначениями, приведенными в работе [3]. Выделим ряд необходимых понятий, используемых в данной работе:

- Множество модулей;
- Множество разделов;
- Множество вычислительных задач, для каждой из которых определены максимальная длительность выполнения задачи, период, раздел, ядро, к которому она привязана;
- Множество работ;
- Зависимости между задачами, определяющиеся ациклическим ориентированным графом;
- Расписание, включающее в себя:
  - множество окон; для каждого из которых определены:
    - время открытия окна;
    - время закрытия окна;
    - модуль, к которому привязано окно;
  - привязка разделов к ядрам;
  - привязка разделов к окнам.

Входными данными для задачи построения статико-динамического расписания с обратной связью являются:

- Исходный алгоритм построения множества окон на основе привязки разделов к ядрам, множества работ и зависимостей между задачами.

- Вычислительная модель выполнения расписания, позволяющая получить временную диаграмму работы системы, т.е. множество интервалов времени, в которые происходит выполнение работ, переключения контекста, паузы и т.п. Для каждого интервала определены тип, время начала интервала, время начала интервала, время конца интервала, ядро, окно, к которому принадлежит интервал;
- Расписание, при выполнении которого не все работы успевают выполниться в срок.

Критерий оптимальности расписания такой же, как в работе [3]. Пытаемся минимизировать количество работ, не успевших выполниться в рамках своих директивных интервалов:

Выходные данные:

- Новая привязка разделов к окнам;
- Новая привязка разделов к ядрам;
- На основе новой привязки разделов к ядрам строится новое расписание в результате работы алгоритма построения расписания.

Новая привязка разделов к окнам получается в результате работы алгоритма с обратной связью, который на основе анализа временной диаграммы осуществляет в общем случае перепривязывание разделов к ядрам, а также перепривязывание разделов к окнам.

Ограничения корректности

Приведем ограничения на расписание, касающиеся построения окон и работы модели, представленные в работе [3]:

1. Окна, к которым привязан раздел, должны быть привязаны к тому же модулю, что и раздел.
2. Разделы, размещенные на одном и том же ядре, не могут быть привязаны к одному и тому же окну.
3. Окна, привязанные к одному модулю, не пересекаются;
4. Множество интервалов, в котором выполняется работа, не выходит за пределы директивного интервала работы.
5. Работа выполняется только в рамках окон, к которым привязан её раздел, с учетом всех переключений контекста;
6. Интервалы выполнения работ одного раздела не пересекаются;
7. Работа начинает выполняться не раньше получения данных от работ, от которых она зависит;
8. Работа начинает выполняться не раньше завершения более приоритетной работы того же раздела, начавшейся раньше.

Примечание: ограничения 1-3 учитываются алгоритмом, а ограничения 4-8 – вычислительной моделью.

Приведем также ограничения на привязку разделов к ядрам, изложенные в работе [1]:

9. Каждый раздел должен быть привязан не более чем к одному ядру;

10. Каждый раздел может быть привязан только к тем ядер, которые разрешены для этого раздела в наборе входных данных;

11. Не должен быть превышен верхний предел загрузки каждого ядра вычислительными задачами раздела;

Согласно [3] алгоритм построения расписания является корректным, т.е результатом его работы является расписание, удовлетворяющее всем ограничениям корректности 1-8. В основе алгоритма с обратной связью используются функции:

- Перепривязка разделов к ядрам.
- Перепривязка разделов к окнам.

Алгоритм с обратной связью обязан оперировать только такими действиями, которые бы не нарушали ограничения 1-3 и 9-11, поскольку ограничения 4-8 учитываются моделью.

Ряд ограничений 1-11 на корректируемое расписание будет соблюден, если будут соблюдены ограничения корректности 9-11 на привязку разделов к ядрам для алгоритма. Также корректировка перепривязки разделов к окнам должна удовлетворять соответствующему ряду ограничений 1-3.

Данные ограничения на корректируемое расписание учитываются в процессе работы алгоритма и не нарушаются.

Анализ задачи

В ходе анализа выполнения расписания были выявлены причины, которые могут приводить к нарушению ограничений корректности расписания:

1. Перегрузка ядра. Слишком много разделов распределено на ядро, в результате чего работы просто не успевают выполниться. В загрузку ядра не включаются затраты на переключения окон, с их учетом загрузка могла и превысить 100%. Этот момент не учитывается на этапе распределения разделов по ядрам. Проблема решается более равномерным распределением нагрузки по модулям.

2. Локальная перегрузка. По каким-то причинам на определенные отрезки времени приходится слишком много работ. В результате часть работ не выполняется в требуемое время, хотя в других частях расписания может быть "вакуум" и средняя загрузка ядра

оказывается не такой уж и большой. Это можно вылечить разведением конфликтующих задач по разным ядрам.

3. Слишком длинные цепочки работ. Одни работы могут ожидать сообщений от других, и далее по цепочке. Если длина этой цепочки слишком большая, последние работы могут просто не успеть. Цепочки можно сократить при помощи перемещения задач на один модуль - передача сообщений через память быстрее, чем через канал.

4. Отсутствие доступного окна. В рамках периода работы может не найтись подходящего окна для ее выполнения, хотя общая загрузка ядра, приходящаяся на этот отрезок времени, позволяет работе быть выполненной. Можно изменить значение раздела на другом свободном окне, находящегося в рамках интервала периода работы так, чтобы в нем нашлось время для выполнения опоздавшей работы из этого раздела.

В результате анализа временной диаграммы выполнения расписания на имеющихся тестовых начальных данных было выявлено, что основной причиной опоздания большинства работ является локальная перегрузка. Также ряд работ опаздывали из-за отсутствия подходящего доступного окна.

Слишком длинные цепочки работ могут также оказывать влияние на наличие опоздания работ, но данный вопрос требует отдельного исследования, которое является перспективным направлением дальнейшей работы.

Описание алгоритма

Процедура перепривязки разделов к окнам

1. Изначально множество соотношений раздел-окно для привязки пусто.
2. Для каждой из опоздавших работ определить, может ли она быть выполнена за счет перепривязки раздела к другому окну. Для этого:
    1. Выбрать интервалы на ядре в рамках периода опоздавшей задачи.
    2. Если такой интервал имеется, то проверить для него наличие выполнения условий:
        - не является загруженным интервалом;
        - длительность интервала не превышает длительность выполнения работы на этом ядре;
    3. Если такой интервал имеется, то проверить для него наличие выполнения условий:

- окно данного интервала не содержит интервалов выполнения работ;
- окно данного интервала не было ранее использовано для переприв язки раздела;

4. Если все условия соблюдаются, то работа может быть выполнена за счет переприв язки раздела к окну. Фиксируем возможность привязки.

3. Если все работы из множества могут быть выполнены за счет переприв язки разделов по окнам, то выполнить эту переприв язку.

Процедура переприв язки разделов к ядрам

1. Запрещаем переприв язывать разделы на ядра с опоздавшими задачами.
2. Выбираем задачи в порядке уменьшения количества экземпляров опоздавших работ.
3. Выбрать ядро, на которое переприв язать раздел очередной задачи. Для этого:
   1. Выбрать список ядер, доступных для привязки раздела. Для этого в список доступных ядер включить те ядра, для которых выполняются все условия:
      - ядро является доступным для привязки к нему раздела;
      - на ядре не было опоздавших работ;
      - ядро не использовалось в алгоритме ранее для переприв язки;
      - загрузка ядра с учетом вклада в загрузку привязываемого раздела не превысит максимально допустимую загрузку ядра;
      - для каждого экземпляра опоздавшей работы рассматриваемое ядро содержит интервалы в пределах периода работы, на которых оно не загружено, суммарной длительностью превышающих длительность выполнения работы.
   2. Выбрать оптимальное для переприв язки раздела ядро. Для этого из списка доступных ядер выбрать те, с которыми загрузка канала сообщениями у данного раздела максимальна. А уже из этого списка выбрать наименее загруженное ядро.
4. Если удалось выбрать ядро, удовлетворяющее всем условиям, то:
   1. Изменить привязку раздела к этому ядру.

2. Исключить из списка рассматриваемых опоздавших задач текущую задачу.
3. Исключить из списка возможных для привязки разделов текущее ядро.
4. Перейти к п.3 и рассмотреть очередную опоздавшую задачу.

5. Иначе завершить переприв язку разделов по ядрам.

Алгоритм с обратной связью

Алгоритм с обратной связью построения статико-динамического является итерационным. Условием остановки алгоритма является превышение заданного количества итераций или же условие успешного построения корректного расписания.

Каждая итерация состоит из следующих шагов:

1. Построение расписания.
2. Построение временной диаграммы при помощи модели.
3. Процедура переприв язки разделов к окнам. Если все работы возможно выполнить за счет переприв язки разделов по окнам, то проделываем это. После заново строим модель (п.2) без необходимости перестроения расписания и убеждаемся, что все работы выполняются согласно модели.
4. Процедура переприв язки разделов к ядрам.

Иначе остается произвести переприв язку разделов по ядрам. После этой итерации вернуться на п.1, т.е перестроить расписание, построить модель и опять провести ее анализ.

Вычислительная сложность алгоритма

В общем случае сложность алгоритма анализа модели складывается из сложности алгоритма переприв язки разделов к окнам и алгоритма переприв язки разделов к ядрам. Она равна:

O (N_F*N_I*(N_c+ (N_I)+N_c*N_p) , где

- N_F - количество опоздавших работ;
- N_I- количество всех интервалов на временной диаграмме;
- N_c- количество всех ядер;
- N_p- количество всех разделов.

Стоит отметить, что алгоритм анализа модели имеет значительно меньшую вычислительную сложность по сравнению с сложностью алгоритма построения расписания или алгоритма построения модели.

Экспериментальная апробация

Целью экспериментов было выяснить, позволяет ли алгоритм с обратной связью строить корректное расписание. Реализация алгоритма была протестирована на разных исходных данных и разных вариантах параметров расписания, таких как максимальная загрузка ядра временным разделом, максимальная загрузка ядра. Исходные данные включают:

- S1 – данные реальных систем авиационного назначения;
- S2 – искусственно сгенерированные данные, где количество задач и сообщений было увеличено вдвое по сравнению с предыдущим набором исходных данных;

Результаты экспериментов представлены в таблице 1.

Таблица 1

Результаты экспериментов.

входные данные				результаты	
набор данных	P_load,%	C_load, %	N_before	N_after	Ni
S1	20	70	2	0	2
S2	20	70	140	0	8
S1	10	40	4	0	4
S1	20	20	17	0	1
S2	10	40	67	30	10
S2	30	60	28	6	10
S2	20	50	67	0	2
S2	10	70	46	0	1

P_load - максимальная загрузка ядра временным разделом;

C_load - максимальная загрузка ядра;

N_before - количество опоздавших работ в исходном расписании;

N_after - количество опоздавших работ в итоговом расписании;

Ni - количество итераций;

Время анализа модели и перепривязки разделов можно считать мгновенным по сравнению со временем перестроения расписания после перепривязки разделов. На практике время работы алгоритма с обратной связью можно считать как суммарное время построения расписания и время построения модели, помноженное на количество итераций.

Из результатов экспериментов следует, что разработанный алгоритм с обратной связью во всех рассмотренных случаях уменьшает количество опоздавших работ и в большинстве случаев строит полные корректные расписания.

Таким образом, в результате апробации алгоритма на тестовых данных было выявлено, что алгоритм может использоваться на практике и разрешать проблему построения статико-динамического расписания, при этом алгоритм не может гарантировать успешность построения корректного расписания.

Заключение

Перспективным направлением дальнейшей работы является исследование и учет влияния слишком длинных цепочек работ как причины опоздания работ для алгоритма с обратной связью.

Литература

1. Balashov V., Balakhanov V., Kostenko V. 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. Arinc Specification 653. Airlines Electronic Engineering Committee [PDF] (http://www.arinc.com).

3. С. В. Селецкий Алгоритмы планирования вычислений в системах на основе интегрированной модульной авионики // Дипломная работа. Москва, ВМК МГУ, 2015. [PDF] (https://cloud.lvk.cs.msu.su/index.php/s/WZtuJPhGzD1AAVc)

Запутляев И.А., Волканов Д.Ю.

Исследование адаптивного генетического алгоритма для решения задачи оптимизации надёжности распределённых вычислительных систем[1]

Введение

Распределённая вычислительная система (РВС) – вычислительная система, узлы которой распределены в пространстве [1]. Выделяют четыре фундаментальные характеристики РВС: функциональность, производительность, надёжность и стоимость [2]. Под надёжностью РВС будем понимать вероятность её безотказной работы в течение некоторого заданного промежутка времени [3]. Функциональность системы задается с помощью спецификаций, а её стоимость в реальных проектах обычно ограничена сверху определённой константой. Таким образом, возникает задача разработки такой РВС, которая будет иметь заданную функциональность и максимальную надёжность при ограничении на стоимость.

Надёжность вычислительной системы можно повысить двумя способами: использование более надёжных компонентов при построении системы или использование механизмов обеспечения отказоустойчивости (МОО).

В работе рассматриваются следующие МОО:

1) N-версионное программирование (NVP). Этот подход заключается в том, что есть N компьютеров (N – нечётное), выполняющих параллельно одни и те же вычисления, и есть модуль голосования, сравнивающий результаты, и принимающий окончательное решение, если большинство компьютеров дало одинаковый результат. В работе рассматриваются 2 варианта NVP при N = 3. В NVP/0/1 все 3 программные компоненты работают на одном аппаратном компоненте. Подход не предусматривает аппаратных неисправностей, но устойчив к одной программной неисправности. В NVP/1/1 программные компоненты работают на разных аппаратных компонентах. Подход устойчив к одной аппаратной неисправности, либо к одной программной неисправности.

2) Восстановление блоками (RB/1/1). В этом случае имеется N программных компонент, M аппаратных, а также модуль принятия решений (контрольный тест). Сначала на всех аппаратных компонентах запускается первая программная версия. После завершения работы первой версии контрольный тест принимает решение или нет. В этом

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

случае происходит возврат в начало и запуск второй программной версии и так далее. Процесс продолжается до тех пор, пока результат одной из версий не будет принят, либо результат работы всех версий будет отклонен контрольным тестом. В данной работе используется по 2 аппаратных и программных компонента (RB/1/1). Такой подход устойчив к одной аппаратной и к одной программной неисправности.

1. Постановка задачи

В данной работе используется формализация задачи из работы [4]. Структура РВС задаётся в виде связанных между собой модулей. Для каждого модуля заданы доступные МОО. Каждый модуль содержит не менее одного аппаратного и одного программного компонентов. Известны наборы возможных вариантов устройств и программ, которые могут присутствовать в модуле в качестве соответственно аппаратного и программного компонентов. Для каждого варианта устройства или программы известны надёжность и стоимость.

Необходимо выбрать такой набор устройств и программ, при которых надёжность системы максимальна с учётом ограничения на стоимость.

Пусть заданы следующие характеристики системы:

- n – количество модулей РВС;
- $p_i, q_i, \forall i \in [1,n]$ – количество доступных версий соответственно аппаратного и программного компонентов в i-ом модуле;
- R_{ij}^{hw}, $C_{ij}^{hw}, \forall i \in [1,n]$, $\forall j \in [1,p_i]$ – надёжность и стоимость j-ого варианта аппаратного компонента i-ого модуля;
- R_{ij}^{sw}, $C_{ij}^{sw}, \forall i \in [1,n]$, $\forall j \in [1,q_i]$ – надёжность и стоимость j-ого варианта программного компонента i-ого модуля;
- FT_i, $\forall i \in [1,n]$ – множество доступных вариантов МОО i-ого модуля; $FT_i \subseteq \{NVP/0/1,\ NVP/1/1,\ RB/1/1,\ None\}$;
- P_{rv}, P_{all}, P_d – вероятность отказа между двумя версиями программного компонента, вероятность одновременного отказа всех версий программного компонента и вероятность отказа модуля голосования соответственно;
- $C_{system}^{\max}$ – максимальная допустимая стоимость РВС.

Используя R_{ij}^{hw}, R_{ij}^{sw}, P_{rv}, P_{all} и P_d, можно вычислить надёжность каждого модуля для любой конфигурации системы по формулам из [5]. Надёжность системы R_{system} равна произведению

надёжностей всех её модулей. Стоимость системы C_{system} можно вычислить, используя C_{ij}^{hw} и C_{ij}^{sw}.

Требуется определить конфигурацию $System_{best} \in Systems$ ($Systems$ – множество всех возможных конфигураций РВС), такую что:

$$R_{System_{best}} = \max_{Systems} R_{system},$$

при условии:

$$C_{System_{best}} \le C_{system}^{\max}.$$

то есть стоимость системы не должна превышать максимально допустимой.

2. Существующие методы решения задачи

Данная задача решалась с помощью Генетического Алгоритма (ГА) [4], Гибридного Генетического Алгоритма (ГГА) с блоком нечёткой логики [3,4], Алгоритма Имитации Отжига [6], Алгоритма Поиска с Запретом [7]. Обзор методов решения данной задачи приведен в [11].

В данной работе для решения поставленной задачи предлагается Адаптивный Генетический Алгоритм (АГА), являющийся модификацией классического Генетического Алгоритма. Главной его особенностью является способность изменять свои настройки во время работы, что позволяет находить лучшие по качеству решения. Это возможно благодаря методам адаптации алгоритма к рельефу целевой функции.

Рассмотрим схему работы Генетического Алгоритма (ГА):

1. Установка начальных параметров алгоритма.
2. Генерация случайным образом начальной популяции решений.
3. Вычисление целевой функции для каждого из решений, а также проверка ограничения на стоимость; фиксация лучшего на текущий момент решения.
4. Выполнение операции селекции по схеме пропорционального отбора.
5. Отбор особей для скрещивания в отдельную промежуточную популяцию: популяция сортируется и отбираются лучшие N_{cross}% особей по значению целевой функции.
6. Выполнение операции одноточечного скрещивания; для каждой пары родительских решений вероятность скрещивания равна P_{cross}.

7. Формирование новой популяции: в неё попадает N_{cross} % лучших по значению целевой функции решений, полученных в результате скрещивания, и (100 – N_{cross}) % лучших решений из текущей популяции.

8. Выполнение операции одноточечной мутации: случайное изменение одного из компонентов решения; мутирует только N_{mut} % худших по качеству решений с некоторой вероятностью P_{mut}.

9. Проверка критерия останова: если критерий не достигнут, перейти к шагу 2, иначе вывести лучшее решение и завершить работу алгоритма.

В ГГА вместе с вычислением целевой функции для каждого решения из популяции происходит вычисление среднего для популяции значения целевой функции. В зависимости от этих значений изменяются значения параметров алгоритма согласно схеме нечёткой логики. Это нужно для того, чтобы в автоматическом режиме корректировать настройки ГГА, управлять степенью влияния операций селекции, скрещивания и мутации на работу алгоритма согласно некоторым правилам в зависимости от результатов его работы в каждом поколении.

Введём следующие обозначения:

- R_1^{av} и R_0^{av} – средние значения целевой функции в текущей и предыдущей популяциях;
- $R_1^{\max}$ и $R_0^{\max}$ – лучшие значения целевой функции в текущей и предыдущей популяциях.

Изменяемые параметры ГА:

- N_{cross} – процент от популяции лучших особей, которые затем будут скрещиваться;
- P_{cross} – вероятность скрещивания;
- N_{mut} – процент лучших особей текущей популяции, которые не мутируют;
- P_{mut} – вероятность мутации.

Параметры N_{mut} и P_{mut} имеют три значения (большое, среднее и малое). Параметры N_{cross} и P_{cross} имеют два значения (большое и малое). Конкретное числовое значение определяется параметрами оптимизируемой системы.

В зависимости от изменений параметров R^{av} и $R^{\max}$ в процессе работы ГГА происходит переключение значений параметров алгоритма по следующей схеме, изображённой в таблице 1 [4].

Такой подход позволяет повысить качество находимых алгоритмом решений без существенного повышения его вычислительной сложности.

Таблица 1

Правила работы схемы нечёткой логики

	$R_0^{av} < R_1^{av}$	$R_0^{av} \approx R_1^{av}$ (~3%)	$R_0^{av} > R_1^{av}$
$R_0^{\max} < R_1^{\max}$	Малый P_{mut} Малый N_{mut} Большой N_{cross} Большой P_{cross}	Средний P_{mut} Средний N_{mut} Большой N_{cross} Большой P_{cross}	Большой P_{mut} Большой N_{mut} Большой N_{cross} Большой P_{cross}
$R_0^{\max} \approx R_1^{\max}$ (~3%)	Малый P_{mut} Малый N_{mut} Малый N_{cross} Малый P_{cross}	Средний P_{mut} Средний N_{mut} Малый N_{cross} Малый P_{cross}	Большой P_{mut} Большой N_{mut} Малый N_{cross} Малый P_{cross}

3. Адаптивный генетический алгоритм

Идея предложенного в данной работе АГА, такая же как и в ГГА. Она заключастся в изменении параметров мутации P_{mut} и скрещивания P_{cross} решений. Для этого вводятся индивидуальные параметры вероятности и скрещивания для каждого решения из популяции, которые меняются после каждой итерации в зависимости от того, насколько удачным или неудачным оказалось применение конкретной операции к решению.

Индивидуальные вероятности мутации и скрещивания хранятся в векторах M_{mut} и M_{cross} соответственно, где i-ый элемент является значением соответствующего параметра для i-го элемента популяции. Максимально и минимально допустимыми значениями параметров мутации и скрещивания являются 0.9 и 0.1 соотвстственно.

Схемы измснения параметров задаются методами адаптации [11]. В работе рассматриваются следующие методы:

1. **Абсолютный**: задаётся параметр $d \in (0,1)$ изменения значения элементов векторов вероятностей мутации и скрещивания. Если после операции мутации решения, имеющего в популяции номер j, значение целевой функции уменьшилось, то элемент m_j^{mut} вектора вероятностей мутации, соответствующий вероятности мутации данного решения, увеличивается на d, иначе – уменьшается на d. Элементы m_i^{cross} и m_j^{cross} векторов вероятностей скрещивания, которые определяют вероятности скрещивания для i-го и j-го решений в популяции, изменяются аналогичным образом:

$$m_i^{mut} = \begin{cases} 0.1, m_i^{mut} < 0.1 \\ m_i^{mut} + d * \mathrm{sgn}(F_i^{old} - F_i^{new}) \\ 0.9, m_i^{mut} > 0.9 \end{cases}, \quad m_i^{cross} = \begin{cases} 0.1, m_i^{cross} < 0.1 \\ m_i^{cross} + d * \mathrm{sgn}(F_i^{old} - F_i^{new}) \\ 0.9, m_i^{cross} > 0.9 \end{cases}$$

F_i^{old} и F_i^{new} – соответственно значения целевой функции решения до и после применения к нему операции. Использование данного метода позволяет реализовать механизм самообучения, благодаря режиму поощрения операции и режиму наказания операции [8].

2. **Относительный**: позволяет изменять значения элементов векторов вероятностей на переменную величину в зависимости от степени изменения значения целевой функции данного решения после применения операции мутации или скрещивания:

$$m_i^{mut} = \begin{cases} 0.1, m_i^{mut} < 0.1 \\ m_i^{mut} * \dfrac{F_i^{old}}{F_i^{new}} \\ 0.9, m_i^{mut} > 0.9 \end{cases} \quad m_i^{mut} = \begin{cases} 0.1, m_i^{cross} < 0.1 \\ m_i^{cross} * \dfrac{F_i^{old}}{F_i^{new}} \\ 0.9, m_i^{cross} > 0.9 \end{cases}$$

Данный метод делает процесс самообучения генетического алгоритма более чувствительным к изменению качества решения. [8]

3. **Относительный с забыванием**: рассмотренные выше способы коррекции запоминают и хранят предыдущий опыт алгоритма, начиная с первой итерации. Но в ситуации, когда решение перемещается из одного экстремума в другой, необходимо помнить информацию только о текущем экстремуме. Для этого вводится параметр запоминания $s \in (0,1)$:

$$m_i^{mut} = \begin{cases} 0.1, m_i^{mut} < 0.1 \\ s * m_i^{mut} * \dfrac{F_i^{old}}{F_i^{new}}, m_i^{cross} \\ 0.9, m_i^{mut} > 0.9 \end{cases} = \begin{cases} 0.1, m_i^{cross} < 0.1 \\ s * m_i^{cross} * \dfrac{F_i^{old}}{F_i^{new}} \\ 0.9, m_i^{cross} > 0.9 \end{cases}$$

Аналогично со скрещиванием. Заметим, что выбор значения параметра запоминания s определяет скорость перестройки матриц вероятности мутации и скрещивания в зависимости от приближения или удаления к экстремумам в пространстве поиска. При значениях параметра s, близких к 1, забывания практически не будет. В тоже время, при малых значениях этого параметра увеличивается вероятность нахождения локального экстремума [8].

4. **Логистический**: принцип работы этого метода основывается на поведении логистической функции $f(x) = \dfrac{1}{1+e^{-x}}$, которая, при рассмотрении отрезка [0,1], максимально быстро растет на выходе из 0, но практически не меняется при приближении к 1. Суть метода

заключается в том, чтобы сильно менять значение вероятности операции для решения, если после применения этой операции значение целевой функции данного решения близко к среднему значению целевой функции по популяции, и менять слабо, если решение стало близко к лучшему решению. Параметрами метода являются $F_{\max}, F_{avg}, F_i^{new}$ – соответственно максимальное и среднее по популяции значения целевой функции, а также значение целевой функции нового решения, $P_{mut_{\max}}$ и $P_{mut_{\min}}$ – максимально и минимально возможные вероятности операции мутации:

$$m_i^{mut} = \begin{cases} \dfrac{P_{mut_{\min}} - P_{mut_{\max}}}{1 + e^{\left(\frac{2(F_i^{new} - F_{avg})}{F_{\max} - F_{avg}} - 1\right)}} + P_{mut_{\max}}, F_i^{new} \geq F_{avg} \\ P_{mut_{\max}}, F_i^{new} < F_{avg} \end{cases}$$

Аналогично со скрещиванием [9].

$$m_i^{cross} = \begin{cases} \dfrac{P_{cross_{\min}} - P_{cross_{\max}}}{1 + e^{\left(\frac{2(F_i^{new} - F_{avg})}{F_{\max} - F_{avg}} - 1\right)}} + P_{cross_{\max}}, F_i^{new} \geq F_{avg} \\ P_{cross_{\max}}, F_i^{new} < F_{avg} \end{cases}$$

5. **Дисперсионный**: действие этого метода основано на вычислении коэффициента вариации $\delta = \dfrac{D(X)}{E(X)}$ для текущей и новой популяций, равного отношению дисперсии популяции к её математическому ожиданию по значению целевой функции. Если коэффициент вариации для новой популяции больше, чем для предыдущей, то значит увеличилось разнообразие популяции и нужно увеличить вероятность скрещивания и уменьшить вероятность мутации; а если уменьшилось – наоборот. Это позволит сохранять в процессе работы алгоритма разнообразие популяции и не застревать в локальном экстремуме [10]:

$$m_i^{cross} = m_i^{cross} * \frac{\delta_{new}}{\delta_{old}}, \; m_i^{mut} = m_i^{mut} * \frac{\delta_{new}}{\delta_{old}}$$

4. Экспериментальное исследование

В рамках экспериментального исследования необходимо было показать зависимость результатов работы АГА от параметров алгоритма, а также сравнить их с результатами работы других алгоритмов. В связи с этим исследование было разделено на два этапа:

1. сравнение качества решений, полученных при помощи разработанного варианта АГА для различных значений параметров, и выбор лучших их комбинаций;

2. сравнение качества и времени получения решений при помощи разработанного варианта АГА с комбинациями параметров, выбранными на этапе 1, ГА и ГГА для каждого вида системных ограничений. Под временем получения решения понимается количество итераций алгоритма, за которое было найдено решение.

В рамках первого этапа были исследованы комбинации следующих параметров АГА:

- размер популяции: 30 особей;
- значения вероятностей мутации P_{mut} и скрещивания P_{cross} принадлежат равномерному распределению на отрезке [0.1, 0.9];
- доля решений, к которым применяются операции мутации и скрещивания: 0.8;
- критерий останова: 30 итераций без изменения лучшего решения;
- настройки методов адаптации:
  - абсолютный: значения параметра $k \in \{0.05, 0.1, 0.15\}$;
  - относительный: без параметров;
  - относительный с забыванием: значения параметра $s \in \{0.25, 0.5, 0.75\}$;
  - логистический: без параметров;
  - дисперсионный: без параметров.

Параметры экспериментальных РВС:

- две системы по 6 и 8 модулей;
- по 5 версий аппаратного и программного компонента в каждом модуле;
- в каждом модуле может использоваться любой из описанных в статье МОО или МОО может отсутствовать;
- значения надёжности аппаратных и программных компонентов принадлежат равномерному распределению на отрезке [0,85; 0,99];
- значения стоимости компонентов выбраны из отрезка [5; 35];
- значения ограничений на стоимость систем: 600 и 660 соответственно.

АГА запускался на описанных системах по 100 раз для каждой комбинации параметров. В этом случае результаты будут статистически обоснованы [6].

В таблице 2 показаны результаты экспериментов.

Таблица 2

Результаты первого этапа исследования

Кол-во модулей	Огр. на стоим.	Метод Ад-ции	Пар-р	Ср. надёжность	Ср. отклонение
6	600	Абс.	0.05	0.803622	0.027507
		Абс.	0.10	0.80396	0.050269
		Абс.	0.15	0.803937	0.03421
		Отн.	-	0.805121	0.036546
		Отн. с. з.	0.25	0.803906	0.039791
		Отн. с. з.	0.5	0.803655	0.036699
		Отн. с. з.	0.75	0.803237	0.046103
		Лог.	-	0.804578	0.026249
		Дисперс.	-	0.803713	0.036264
8	660	Абс.	0.05	0.749581	0.039729
		Абс.	0.10	0.750258	0.040975
		Абс.	0.15	0.750631	0.02193
		Относ.	-	0.750296	0.030397
		Отн. с. з.	0.25	0.750191	0.030825
		Отн. с. з.	0.5	0.749757	0.04622
		Отн. с. з.	0.75	0.74911	0.035225
		Лог.	-	0.750208	0.044189
		Дисперс.	-	0.749139	0.0451

По итогам 1 этапа исследования лучшие результаты показали относительный и логистический методы адаптации. Эти методы нашли лучшие в среднем решения по совокупности двух систем.

Алгоритмы с использованием абсолютного метода с параметрами $d = \{0.1, 0.15\}$ показали близкие по средней надёжности результаты, но второй вариант оказался более стабильным. Вариант абсолютного метода с параметром $d = 0.5$ хуже проявил себя относительно средней надёжности найденных решений.

Среди вариантов относительного с забыванием метода лучший по обоим параметрам результат показал вариант с параметром $s = 0.25$.

Дисперсионный метод адаптации по обоим параметрам уступил логистическому методу.

В результате, для дальнейшего исследования были выбраны следующие методы адаптации:

- абсолютный с параметром $d = 0.15$;
- относительный;

• относительный с забыванием с параметром $s = 0.25$;

• логистический.

В рамках второго этапа было проведено сравнение с ГА и ГГА, заданными следующими параметрами:

Параметры ГА:

• размер популяции: 30 особей;

• $P_{mut} = 0.75$, $P_{cross} = 0.75$;

• $N_{mut} = 0.8$, $N_{cross} = 0.8$;

• критерий останова: 30 итераций без изменения лучшего решения.

Параметры ГГА:

• размер популяции: 30 особей;

• начальные значения параметров: $P_{mut} = 0.75$, $P_{cross} = 0.75$; $N_{mut} = 0.8$, $N_{cross} = 0.8$;

• $P_{mut} = \{0.9, 0.75, 0.5\}$, $P_{cross} = \{1.0, 0.75, 0.5\}$;

• $N_{mut} = \{0.9, 0.8, 0.7\}$, $N_{cross} = \{0.9, 0.8, 0.7\}$;

• схема нечёткой логики описана в разделе "Существующие методы решения задачи";

• критерий останова: 30 итераций без изменения лучшего решения.

Параметры экспериментальных РВС:

• системы по 12 модулей с параметрами, описанными выше;

• значения ограничений на стоимость систем можно вычислить по формуле: $\max Val - (\max Val - \min Val) * \frac{ConstraitPercent}{100\ \%}$ и разделить на 5 классов в соответствии с параметром *ConstraitPercent*. Параметры *maxVal* и *minVal* – максимальное и минимальное значения стоимости системы среди всех возможных конфигураций.

В таблице 3 представлены средние результаты исследования по всем рассматриваемым системам:

Таблица 3

Результаты второго этапа исследования

Алгоритм	МА	Пар-р	Ср. надёжность	Ср. отклонение	Ср. кол-во итераций
АГА	Абс.	0.15	0.663754	0.030171	217
	Отн.	-	0.665417	0.020151	224
	Отн. с. з.	0.25	0.664783	0.022986	204
	Лог.	-	0.664864	0.020259	215
ГА	-	-	0.656996	0.035588	168
ГГА	-	-	0.661258	0.030753	172

В результате проведенного исследования были сделаны следующие выводы:

- АГА с относительным методом адаптации показал лучшие в среднем результаты на всех рассматриваемых системах;
- логистический и относительный с забыванием методы показали близкие результаты по средней надёжности, обогнав ГА, ГГА и АГА с абсолютным методом;
- абсолютный метод показал близкие результаты с ГГА;
- модификации ГА находят более качественные решения, чем классический ГА вне зависимости от класса системных ограничений;
- ГА и ГГА находят решение за меньшее число итераций, чем варианты АГА. Это происходит из-за того, что ГА и ГГА раньше попадают в ситуацию, когда не могут выйти из локального экстремума и улучшить значение целевой функции. АГА способен эффективно исследовать пространство поиска гораздо дольше;

При сравнении качества полученных решений классическим ГА, ГГА и различными вариантами АГА следует учитывать, что ГА и ГГА изначально запускались на лучших, заранее подобранных параметрах, в то время как АГА запускался на случайных параметрах, в результате чего на старте находился в невыгодном положении. Показана способность АГА к самообучению с целью лучше оптимизировать целевую функцию при использовании рассмотренных методов адаптации.

Заключение

В ходе выполнения данной работы была рассмотрена задача оптимизации надёжности и методы её решения и получены следующие результаты:

- разработан и реализован АГА;
- исследована зависимость решений, найденных АГА, от его параметров;
- выделены методы адаптации, показавшие лучший результат: относительный метод, относительный метод с забыванием и логистический метод.

Возможными направлениями дальнейших исследований являются:

- исследование других методов самообучения и модификаций Генетического Алгоритма;
- рассмотрение варианта многокритериальной оптимизации (например, оптимизировать одновременно надёжность и стоимость);
- исследование других эвристических алгоритмов (например, алгоритмы муравьиных и пчелиных колоний).

Литература

1. Stankovic J. A. et al. Real-time computing //Byte, – 1992. – p. 155-162.

2. Chen L., Avizienis A. N-version programming: A fault-tolerance approach to reliability of software operation //Digest of Papers FTCS-8: Eighth Annual International Conference on Fault Tolerant Computing. – 1978. – p. 3-9.

3. Пашков В.Н. Исследование эффективности модификаций генетического алгоритма для задачи выбора сбалансированного набора механизмов отказоустойчивости в вычислительных системах. Дипломная работа, Москва: МГУ, 2010 [DOC] (https://cloud.lvk.cs.msu.su/index.php/s/ys0jR4dwjlJbWXC).

4. Волканов Д.Ю. Метод сбалансированного выбора механизмов обеспечения отказоустойчивости для распределённых вычислительных систем // Моделирование и анализ информационных систем. - 2016.- Т. 23, № 2. - с. 119-136.

5. Wattanapongskorn N., Coit D. W. Fault-tolerant embedded system design and optimization considering reliability estimation uncertainty //Reliability Engineering & System Safety. – 2007. – v. 92. – №. 4. – p. 395-407.

6. Тимофеев К.В. Разработка и реализация алгоритма имитации отжига для задачи выбора модулей встроенной вычислительной системы реального времени с учётом требований надёжности. Курсовая работа, Москва: МГУ, 2014 [PDF] (https://cloud.lvk.cs.msu.su/index.php/s/BNE1EssEzi4uaOa).

7. Запутляев И.А. Исследование алгоритма поиска с запретом для задачи выбора модулей распределённой вычислительной системы реального времени с учетом требований надёжности. Курсовая работа, Москва: МГУ, 2014 [PDF] (https://cloud.lvk.cs.msu.su/index.php/s/NoGqJq7FHgAzB4K)

8. Костенко В.А., Фролов А.В. Генетический алгоритм с самообучением //Известия РАН. Теория и системы управления, 2015, № 4, с. 24–38.

9. Ma Y., Wan J. Improved hybrid adaptive genetic algorithm for solving knapsack problem //2011 2nd International Conference on Intelligent Control and Information Processing. – 2011.

10. Xie G., Zhang J., Li J. Adapted genetic algorithm applied to slope reliability analysis //Natural Computation, 2008. ICNC'08. FourthInternationalConferenceon. – IEEE, 2008. – v. 1. – p. 520-524.

11. Запутляев И.А. Исследование адаптивного генетического алгоритма для решения задачи оптимизации надёжности. Курсовая работа //Московский государственный университет имени М.В. Ломоносова, 2016. [PDF] (https://cloud.lvk.cs.msu.su/index.php/s/T3IGK7hI0Mx6ZTR)

Раздел IV.

Прикладные аспекты теории чисел для современных вычислителей

Рябов Г.Г.

Генетический взгляд на симметрии простых чисел в структуре натуральных

Введение

Интенсивные исследования в последнее время фундаментальных проблем теории чисел с позиций алгебраической и числовой комбинаторики [2,3,4] не в последнюю очередь связаны с надеждой разработать новые конструктивные инструменты для решения проблем вычислимости, а вместе с этим и новый взгляд на контуры архитектуры будущих компьютеров. Неслучайно с этими исследованиями связаны целые интернациональные группы специалистов (Polymath, Magma, Pari и др.) ориентированные на всесторонний охват программным обеспечением этих и смежных областей. В данной краткой статье делается попытка нащупать контуры вероятного развития в одном из направлений с глубоким взаимопроникновением различных областей математики. Речь идет структуре, объединяющей структуры кратчайших путей в n-кубе, глобальных k-арных деревьев и натуральных N. Все это в целом в русле концепции Ю.И.Манина по созданию конструктивных миров.[1]

1. Шаги по конструированию композиции

Прежде покажем последовательность шагов по конструированию композиции такой структуры.

1. В n-кубе введены биекция k-граней в виде слов A_n над алфавитом A={0,1,2}, алгебра над троичными словами и метрика Хаусдорфа-Хэмминга, вычисляемая на словах.[5,6]

2. Установлена биекция кратчайших k-путей между антиподальными вершинами n-куба и троичными символьными n×(n-k+1)матрицами со строками-словами из A_n,(далее TSM)[7].

3. TSMD-множество матриц диагонального вида-биекции кратчайших k-путей между вершинами (00…0) и (11…1) n-куба. На TSMD определена база размерности k и декомпозиция рекурсии [8].

4. Введено отображение TSMD в вершины глобального k-арного дерева GKT (степень корня k, все остальные вершины-«родитель-k-

детей») – генетическая составляющая в композиции структур. В соответствии с автоморфной функцией для каждой TSMD, однозначно вычисляется ее номер в последовательности натуральных N и тем самым нумерующей вершины GKT.[9]

5. Предложено представление последовательности натуральных как цепей k-кортежей в структурах TSMD-GKT-N для заданного k.

Так цепи k-кортежей натуральных для k=3,5,7,...(первых трех нечетных простых):

T(3)={<1,2,3><4,5,6><7,8,9>...}; (рис.1)

T(5)= {<1,2,3,4,5><6,7,8,9,10><11,12,13,14,15>...};

T(7)={<1,2,3,4,5,6,7><8,9,10,11,12,13,14><15,16,17,18,19,20,21>...};...

T(11);T(13);T(17);...

Каждое натуральное n_s для каждого T(k=p_i), где $p_i \in$ P-множество нечетных простых, имеет однозначную позицию k-кортежа (1,2,3,...N) и однозначную позицию внутри k-кортежа (1,2,...k).

Каждый уровень(поколение) g в k-арном дереве содержит k^g кортежей (корень дерева имеет уровень 0).

Таким образом реализуется связка *«математическая логика - теория представлений - метрическая геометрия – комбинаторика - теория чисел» в рамках одной структуры.*

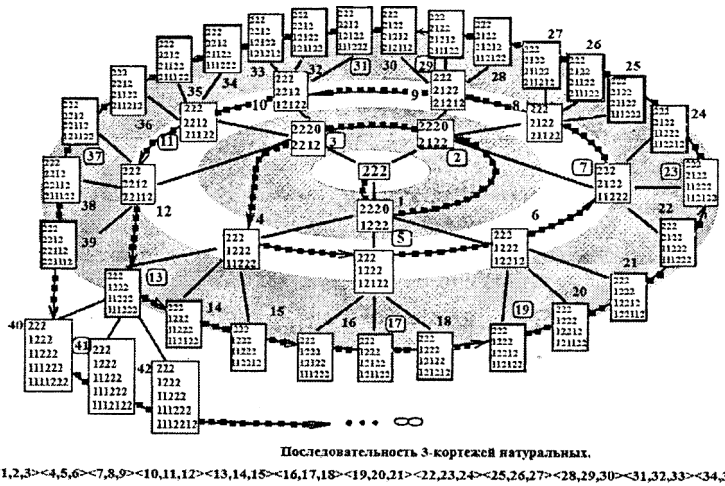

Рис. 1. Композиционная структура STMD-GKT-N для k=3 и уровней g=0-(3-путь в 3-кубе=3-куб);g=1-(3-пути в 4-кубе); g=2-(3-пути в 5-кубе);g=3-(3-пути в 6-кубе); нулевые элементы в матрицах опущены. Натуральные числа рядом с матрицами (и вершинами GKT)-результат действия автоморфной функции для каждой STMD. Простые числа выделены рамкой. Ребра в GKT-отрезки прямых, бесконечная спиральная последовательность натуральных отмечена жирным пунктиром

2. О симметричности простых

Эскизно покажем, что дает эта композиционная структура при рассмотрении одного из аспектов давно интригующей проблемы о структуре простых чисел в структуре натуральных. Более конкретно о симметричности простых относительно натуральных. В качестве индикатора взаимных позиций простых и натуральных рассматривается разностный DT таблоид PxN, каждая клетка(p_i,n_s) которого предназначена для хранения разности $d_{si}=(n_s-p_i)$;$d_{si}\in Z$; Так $d_{si}\geq 0$ при $n_s\geq p_i$(левые простые для n_s) и $d_{si}<0$ при $n_s<p_i$ (правые простые для n_s).

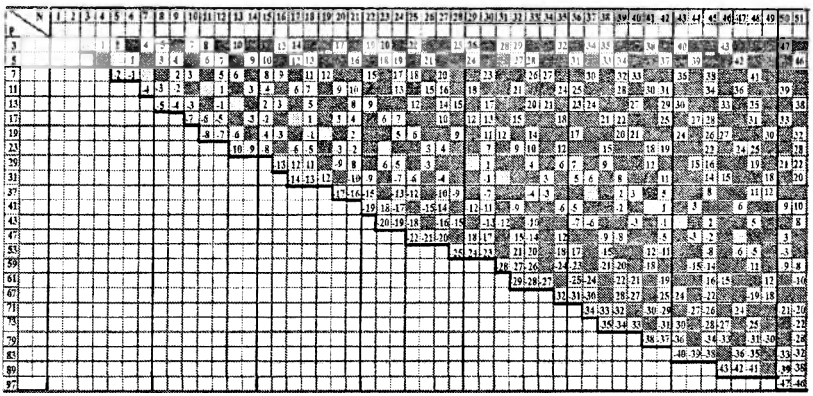

Рис. 2. Вид начального фрагмента разностного таблоида DT. Затемненные клетки отражают отсутствие эквидистантного симметричного простого для каждого натурального

Если $|d_{si}|=|d_{sj}|$, то это означает, что пара эквидистантных от n_s простых p_i и p_j –симметричны относительно натурального n_s. Если для d_{si}(с левым простым p_i) эквидистантная пара не соответствует правому простому p_j (т.е. соответствует составному числу), то левое простое p_i не имеет вообще симметричного правого. Оказывается, что факт отсутствия симметричного правого простого, можно установить без непосредственного вычисления разностей, когда $d_{sj}<0$.

Анализ взаимных позиций в k-кортежах для « левого простого p_i и натурального n_s» и возможного кандидата на эквидистантного правого простого p_j приводят к следующему.

Обозначим позицию левого простого в k-кортеже через x1 и позицию натурального в k-кортеже через x2. Тогда позиция правого эквидистантного от x2 (обозначим через x3, т.е. |x2-x1|=|x2-x3|) может соответствовать простому только *когда позиции x_1 и x_2 не являются решениями линейного сравнения $x_1-2x_2\equiv 0(mod\ k=pi);(1)$*

Это утверждение можно рассматривать как *критерий несовместности позиций в кортежах для пары симметричных простых относительно данного натурального.*

Множество решений (1) для p_i будем обозначать $H(p_i)$ и приведем несколько частных решений для p_i=3,5,7,11 и его общий вид.

H(3)={(x_1=1,x_2=2),(x_1=2,x_2=1),(x1=3,x2=3)};

H(5)={(1,3),(2,1),(3,4),(4,2),(5,5)};

H(7)={(1,4),(2,1),(3,5),(4,2),(5,6),(6,3),(7,7)};

H(11)={(1,6),(2,1),(3,7),(4,2),(5,8),(6,3),(7,9),(8,4),(9,10),(10,5),(11,11)}

Положив p_i+1/2=m, имеем:

$H(p_i)$={(1,m),(2,1),(3,m+1),(4,2),…(p_i-2,p_i-1),(p_i-1,m-1),(pi,pi)}; $|H(p_i)|=p_i$; (2)

Последовательно (вдоль P) пользуясь этим критерием можно в DT затенять клетки, не имеющие парных эквидистантных простых (для сколь угодно больших натуральных), как это уже выполнено на рис.2. Этот процесс прогноза «антисимметричности» вполне детерминирован.

3. Краткая дискуссия о простых

Одним из широко применяемых конструктивных методов в изучении поведения простых является метод, предполагающий некий случайный процесс или процессы, лежащие в основе появления простых среди натуральных. Так в июне 2015 года на международном семинаре «Глобус» доклад профессора Кевина Форда так и назывался «Простые играют в кости?» Возвращаясь к результату из предыдущего раздела рассмотрим отображение решений $H(p_i)$ сравнения (1) для каждого p_i на квадратную p_ixp_i доску. Каждой паре несовместных позиций(x_1,x_2) однозначно соответствует клетка на этой доске, будем считать ее черной. Тогда общая картина решений сравнения (1) (запретных для симметричных простых) будет иметь вид, представленный на рис.3.

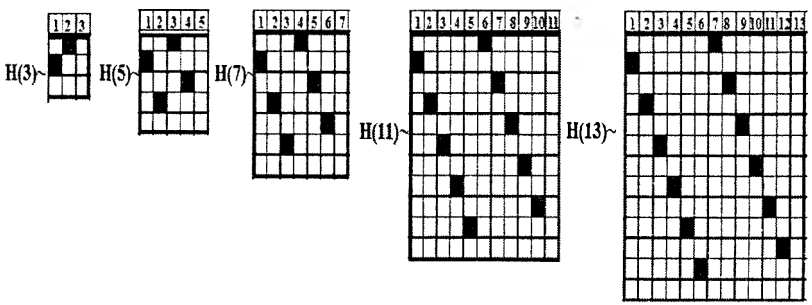

Рис. 3. Отображение решений сравнения (1) на квадратные p_ixp_i доски. Две линии ходом коня

Это отображение склоняет к ответу «Если простые играют, то скорее не в кости, а в шахматы на $p_i x p_i$ досках одной из наиболее экзотических шахматных фигур-конем.

Заключение

Композиции инфинитарных или близких к ним структур (на подобие эскизно рассмотренной выше) вероятно будут играть в будущем все более важную роль в решении биологических, экономических и социальных проблем. Прежде всего в изучении вопросов эргодического поведения одной структуры внутри другой, вплоть до минимальных представлений на уровне диаграмм и таблиц Юнга. Поэтому вопрос как ответит на это архитектура будущих компьютеров, призванных с помощью сопроцессоров эффективно работать с полиморфными биекциями объектов, где роль точной арифметики больших целых чисел в различных системах счисления, во многом может оказаться определяющим.

Литература

1. Yuri I.Manin. Classical computing, quantum computing and Shor`s factoring algorithm.1999. Available: http://arXiv.org/pdf/quant-ph/9903008pdf.

2. K.Ford, B.Green, S.Konyagin, J.Maynard, T.Tao. Long gaps between primes. Available: http://arXiv:1418.5029v2[math.NT] 6 Apr 2015.

3. Polymath-project. The «bounded» gaps between primes. Available: http://arXiv:1409.8361v1[math CO] 30 Sep 2014.

4. J.Pintz. Patterns of primes in arithmetic progressions. Available: http://arXiv:1509.01564v2[math NT] 7 Sep 2015.

5. Г.Г.Рябов. О четверичном кодировании кубических структур// Вычислительные методы и программирование. 2009. Т. 10. №2. с. 340-347.

6. Г.Г.Рябов. Хаусдорфова метрика на гранях n-куба//Фундаментальная и прикладная математика.2010. Т.16. №1. с. 151-155.

7. G.G.Ryabov, V.A.Serov. On classification of k-dimension paths in n-cube// Applied Mathematics, 2014, vol.5, no.4, pp. 723-727.

8. Г.Г.Рябов, В.А.Серов. «Многомерное метро» и символьные матрицы.// INJOIT 2014,Vol 2,No 11, pp.10-18.

9. Г.Г.Рябов, В.А.Серов. Полиморфизм троичных символьных матриц и генетическое пространство кратчайших путей n-куба.// INJOIT 2015,Vol 3, No 7, pp. 1-11.

Н.В. Голышев

Исследование эффективности факторизации больших чисел на графических процессорах

Введение

Важной частью алгоритма факторизации с помощью общего метода решета числового поля является просеивание. Просеивание – процесс выборки данных, удовлетворяющих некоторым условиям. Входом этапа просеивания является факторная база. Задача просеивания, которая будет рассмотрена в этой статье, имеет модельный характер. Список литературы в конце статьи содержит более детальное описание этапа просеивания[1][2][6][7][8].

Под факторной базой FBбудет пониматься набор троек { r, p, log(p) }, где r –отступ, p – простое число (шаг), log(p) – логарифм этого числа.

Областью просеивания SR будем считать отрезок, ограниченный левой границей, принятой за ноль, и правой границей, принятой за некоторое значение RightBound (который представляет из себя размер области просеивания). Результирующий массив представляет из себя одномерный массив, над которым осуществляется просеивание. Значения этого массива отбираются, исходя из интересующих условий просеивания.

Изначальный алгоритм просеивания на центральном процессоре (далее CPU) выглядит следующим образом. В цикле по всем элементам факторной базы, фиксируется один элемент. В теле цикла, начиная с индекса r результирующего массива, и

двигаясь по нему с шагом p до правой границы области просеивания, осуществляется инкремент массива по текущему индексу на значение log(p).

Результатом просеивания является массив целых чисел, в котором интересующими элементами являются превосходящие некоторое пороговое значение.

```
for (i = 0; i < FB_SIZE; i++)
{
  r = FB[i].r;
  p = FB[i].p;
  logp = FB[i].logp;

  while (r <SR.RightBound)
  {
    A[r] += logp;
    r += p;
  }
}
```

Псевдокод, иллюстрирующий модельный алгоритм просеивания на CPU

При увеличении области просеивания, увеличивается и количество операций записи в результирующий массив. Хотелось бы, чтобы эти операции выполнялись не над всем множеством данных, а над их уменьшенным агрегированным множеством. По возможности, вместо множества хождений по результирующему массиву осуществить один проход по нему, заменив инкременты на однократное присвоение уже подсчитанного значения.

При выборе технологии параллельной реализации алгоритма выбор пал на GPU, который обладает более быстрой в сравнении с CPU памятью. Данная особенность имеет высокую значимость, т.к. в алгоритме, рассмотренном ниже, сложные арифметические операции почти полностью отсутствуют, тогда как работа с большим объёмом данных присутствует на каждом шаге алгоритма. При данной модели алгоритма, использование GPU выглядит более перспективным.

Для реализации параллельного кода на GPU была использована технология CUDA разработки на графических процессорах и библиотека CUB [5] для работы с массивами пар ключ-значение.

1. Задача и алгоритм решения

Предлагается модифицировать алгоритм просеивания для более эффективной реализации его на GPU путём ухода от прямого взаимодействия с результирующим массивом. Для этого, данные будут представлены в виде пар ключ-значение, где ключом будет являться индекс элемента в результирующем массиве, а значением будет являться значение массива по этому индексу. Таким образом получится уйти от одного массива большого объёма и работать с достаточно произвольными подмножествами элементов этого массива. Массив всех пар ключ-значение, применение которого к результирующему массиву будет являться выходом этапа просеивания, будем называть массивом обновлений.

```
// ВычислениемассивовNиOffset
N = ComputePairSequencesSizes(FB, SR);
Offset = ExclusiveSum(N);

// подсчёткол-ваподразмербуфера
chunk = GetChunkSize(N, Offset);

for (k = 0; k < FB_SIZE; k+=chunk)
{
  // копироватькусок chunk на GPU
int[Offset[k+chunk]-Offset[k]] keys;
int[Offset[k+chunk]-Offset[k]] values;

RunKernel ( =>
  {
for (i = 0; ; i++)
    {
seq = blockIdx.x + gridDim.x * i;
```

```
pairCount = N[seq];

r = FB[seq].r +threadID[1] * FB[seq].p;
    p = FB[seq].p * blockDim;
logp = FB[i].logp;

currOffset = Offset[seq] + threadID;
endOffset = currOffset + pairCount;

    while (currOffset<endOffset)
    {
      keys[index] = r;
      values[index] = logp;
      r += p;
currOffset += blockDim;
    }
  }
  }

  (keys, values).SortByKey();
  (keys, values).ReduceByKey();

  Thrust::Increment(A, keys, values) ( =>
    for (j = 0; j <keys.Count; j++)
    {
      A[keys[j]] = values[j];
}
  }
}
```

Псевдокод, иллюстрирующий модельный алгоритм просеивания на GPU

Сгенерируем массив обновлений на основе факторной базы. Для начала, потребуется произвести расчёт количества пар, которые на следующем этапе будут сгенерированы для

каждого элемента факторной базы. Для этого происходит вызов ядра CUDA [10], который ставит каждому элементу факторной базы FB_i в соответствие число N_i количество элементов, рассчитанных по формуле

$$FB \to N : N_i = \frac{SR.RightBound - FB_i * r - 1}{FB_i * p} + 1 \qquad (1)$$

Затем происходит вычисление массива `Offset` отступов на основании полученного массива `N` количества пар с помощью

[1]Для определения координаты нити в блоке и координаты блока в сетке на GPU используются специальные свойства библиотеки CUDA. Так, координата нити в блоке имеет значение threadIdx.x (= threadID), координата блока в сетке блоков blockIdx.x (= blockID),размерность блока blockDim.x (=blockDim), а размерность сетки по оси x - gridDim.x(= gridDim).

`ExclusiveSum()` из библиотеки CUB, которая осуществляет подсчёт средствами GPU. Функция выполняет преобразование вида

$$N \rightarrow Offset : Offset_k = \sum_{i=0}^{k-1} N_i, Offset_0 = 0 \quad (2)$$

Следом идёт вычисление на GPU, сколько пар может быть помещено в буфер, и сама генерация этих пар. Следует отметить, что этот и последующие шаги выполняются в цикле, обрабатывая факторную базу кусками.

Вычисление, сколько пар может быть помещено в буфер, напрямую зависит от аппаратных особенностей системы, на которой происходит исполнение программы. К этому вопросу мы вернёмся более подробно после описания алгоритма.

Для генерации используется размер сетки, равный 1x65535 блоков и 1x128 нитей в блоке. В рамках одного блока в shared памяти фиксируется один элемент факторной базы, который обрабатывается параллельно 128 нитями. В рамках каждой нити фиксируется свой отступ для данного элемента факторной базы, равный сумме исходного отступа элемента факторной базы и номера этой нити в блоке, умноженной на шаг

$$r = r + threadID * p \quad (3)$$

Фиксируется шаг, одинаковый для каждой нити, и равный произведению исходного шага на количество этих нитей в блоке.

$$p = p * gridDimID \quad (4)$$

В каждой нити в цикле по области просеивания, с отступом r и шагом p, происходит генерация пар ключ-значение, где ключом является текущее значение индекса r + p*i. Выход из цикла происходит по достижению индекса правой границы области просеивания.

Каждый блок сетки работает со своим участком памяти на GPU. Вся память на GPU, по сути, и занята массивом под эти пары. В этой связи стоит отметить, что эффективность предложенного алгоритма напрямую зависит от объёма доступной памяти устройства (что будет отображено ниже).

На выходе этапа генерации имеем массив пар ключ-значение, по своему объёму примерно сопоставимым с доступной памятью устройства. Следующим этапом необходимо произвести редуцирование этих пар с целью получения результирующих значений, которые и будут присвоены элементам результирующего массива.

Редуцирование по ключу выполняется библиотекой CUB. Однако, стоит отметить, что лучших результатов удалось достичь, используя сортировку непосредственно перед редуцированием. Для сортировки, опять же, была использована библиотека CUB.

Теперь, после этапа редуцирования, конечный массив пар готов для того, чтобы с его помощью осуществить проход по результирующему массиву и по индексу «ключ» положить значение «значение». Для этого был использован `permutation_iterator` из библиотеки Thrust [9].

Изложенный алгоритм отлично работает на небольших объёмах данных, когда весь массив пар целиком может быть размещён в памяти видеокарты. Однако, ресурсов устройства недостаточно для такого рода задачи. В этой связи, в данный алгоритм была добавлена итеративность, которая сделала возможным обработку большой факторной базы на большой области просеивания кусками. Для этого и был введён шаг вычисления количества пар, которые могут быть помещены в буфер.

На этом шаге, имея массив N и массив отступов *Offset*, вычисляется максимально возможный размер куска из факторной базы, который мог бы поместиться в память устройства.

2. Эксперименты

Все эксперименты проводились при выполнении программы на одном ядре процессора Intel Core i7 4771 и видеокарты GeForce GTX 980 Ti с фиксированным размером буфера на видеокарте в 2гб. Для чистоты эксперимента, все замеры были сделаны на программе, выполнение которой проходило на одном ядре CPU.

Целью экспериментов являлось исследование эффективности реализованного алгоритма на GPU в сравнении с таковым на CPU. Метрикой ускорения выступило отношение времени исполнения программы на CPU к времени исполнения на GPU.

Как было отмечено выше, данный алгоритм чувствителен к объёму доступной памяти. Исходя из результатов проведённых экспериментов, можно сделать вывод о практически линейной зависимости ускорения от объёма памяти, выделенной для решения задачи. Объяснить представленные в таблице 1 значения можно тем, что с увеличением доступной памяти уменьшается кол-во операций обмена данными между хостом и устройством (увеличивается размер куска факторной базы).

Таблица 1

Ускорение в зависимости от объёма доступной памяти

Объём, мб	128		256	512	1024	2048
Ускорение	1.41		1.66	2.17	2.70	3.83

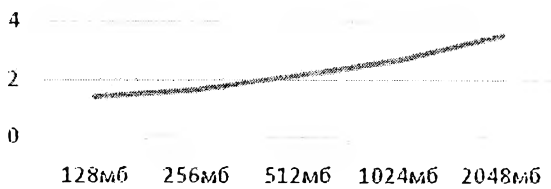

График 1. Ускорение в зависимости от объёма доступной памяти

Данная зависимость алгоритма от объёма памяти может быть использована как для увеличения размерности факторной базы, так и для увеличения области просеивания с целью увеличения эффективности. Дальше будет рассмотрен каждый из этих вариантов.

Следующий эксперимент, результаты которого представлены в таблице 2, показывает зависимость времени исполнения от размера факторной базы при фиксированном значении размера области просеивания.

Таблица 2

Ускорение в зависимости от размера факторной базы

Размер FB	$15*10^6$	$10*10^6$	$5*10^6$	$1*10^6$
CPU, мс	18536	19528	15433	13806
GPU, мс	5421	5356	4723	3646
Ускорение	3,42	3,65	3,27	3,79

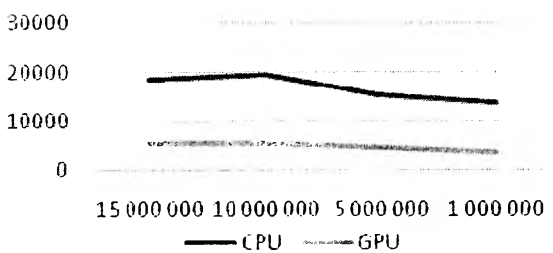

График 2. Ускорение в зависимости от размера факторной базы

На представленном графике видно, что время исполнения не сильно коррелирует с размером факторной базы.

Зависимость ускорения от размера факторной базы прослеживается слабо. Во многом благодаря тому, что для разных размеров, по сути, меняется лишь количество итераций алгоритма,

тогда как его внутренняя структура остаётся постоянной для всех размеров. Это связано с дроблением задачи на блоки по факторной базе.

Таблица 3 показывает результатыэксперимента, отражающего зависимость времени исполнения от размера области просеивания при фиксированном значении размера факторной базы.

Таблица 3

Ускорение в зависимости от размера области просеивания

Размер области	$25*10^8$	$20*10^8$	$10*10^8$	$5*10^8$
CPU, мс	25685	19528	6081	2642
GPU, мс	6954	5356	2024	839
Ускорение	3,69	3,65	3,00	3,15

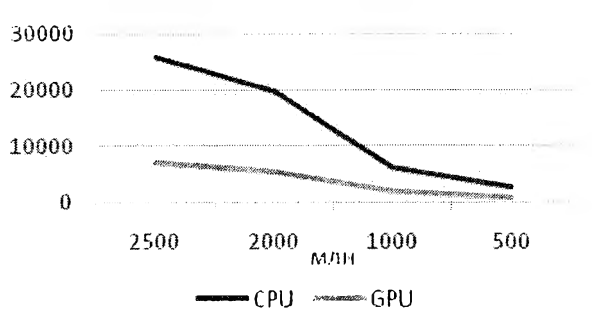

График 3. Ускорение в зависимости от размера области просеивания

Здесь прослеживается более сильная корреляция времени исполнения от размера области, нежели от размера факторной базы. Следует отметить, что при бóльших размерах области получается немного более высокое ускорение. Опять же, это можно объяснить тем, что при бóльших размерах области происходит обмен между хостом и устройством меньшим объёмом входных данных, и GPU дольше по времени занят полезной нагрузкой.

Заключение

В данной статье описан подход к реализации этапа просеивания модельного алгоритма факторизации общего метода решета числового поля. Исследована эффективность полученного алгоритма. Показано, что существует почти линейная зависимость ускорения от размера доступной памяти устройства.

Из возможных направлений для дальнейших исследований можно отметить возможность замены ориентированности на элементы факторной базы, так как более естественным решением было бы сделать

акцент на массив обновлений. Данный подход дал бы более плотную загрузку памяти GPU, чем в предложенном варианте.

Литература

1. A. K. Lenstra The number field sieve [PDF] (http://www.std.org/~msm/common/nfspaper.pdf).

2. Andrea Miele Cofactorization on Graphics Processing Units [PDF] (https://eprint.iacr.org/2014/397.pdf).

3. Репозиторий проекта GLS [HTML] (https://github.com/pstach/gls/).

4. Репозиторий проекта msieve [HTML] (http://sourceforge.net/projects/msieve/).

5. Репозиторий проекта CUB [HTML] (https://nvlabs.github.io/cub/index.html).

6. О. В. Бабуль, О реализации стадии просеивания в алгоритмах решета числового поля, Тр. Ин-та матем., 2009, том 17, номер 1, 19–26.

7. О.Н. Василенко - Теоретико-числовые алгоритмы в криптографии, МЦНМО, 2003.

8. С. А. Желтов - Эффективные вычисления в архитектуре CUDA в приложениях информационной безопасности, 2014.

9. An Introduction to Thrust webinar by NVidia (2014).

10. CUDA C Best Practices Guide [HTML] (http://docs.nvidia.com/cuda/cuda-c-best-practices-guide).

Аннотации

Скобцова Ю.А., Пашков В.Н. Исследование и разработка сетевого приложения для распределенной платформы управления в программно-конфигурируемых сетях // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМиК МГУ, 2016.

Статья посвящена исследованию и разработке сетевого приложения мониторинга состояния контроллера в рамках распределенной платформы управления в ПКС. Данный вид мониторинга необходим для обнаружения перегрузок, отказов на уровне управления программно-конфигурируемой сетью. Приложение мониторинга контроллера реализовано для распределенной платформы управления на базе контроллера RuNOS. В работе приводятся описание реализованного приложения и экспериментальное исследование.

Ил.: 5 рис., Библиогр.: 9.

Сычева Е.А., Пашков В.Н. Исследование и разработка алгоритмов обнаружения и предотвращения DDoS атак на контроллер в программно-конфигурируемых сетях // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМиК МГУ, 2016.

В статье приведена постановка задачи обнаружения и предотвращения DDoS атак на контроллер в программно-конфигурируемых сетях, приведено описание статистического метода предотвращения DDoS атак на контроллер на основе классификации пользователей, описание предложенных методов решения, а также их экспериментальное исследование.

Ил.: 1 рис., Библиогр.: 9.

Хахалин А.С., Чемерицкий Е.В. Разработка метода организации централизованного контроля над сетью программно-конфигурируемых коммутаторов без использования вспомогательной сети управления // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМиК МГУ, 2016.

Статья посвящена разработке метода обеспечения отказоустойчивости управления ПКС с интегрированным управлением. Метод обеспечивает восстановление работоспособности сети после единичных или множественных отказов при условии существования хотя бы одного рабочего пути между коммутатором и контроллером.

Ил.: 2 рис., Библиогр.: 7.

Янбулатов Р.А., Смелянский Р.Л. Об одном подходе к эволюции традиционной сети в программно-конфигурируемую сеть //

Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМиК МГУ, 2016.

В статье рассматривается задача планирования эволюции традиционных сетей (основанных на OSPF или IS-IS) в программно-конфигурируемые сети, предложены алгоритмы выбора маршрутизаторов в сети, которые следует заменить на ПКС коммутаторы в первую очередь.

Ил.: 3 рис., Библиогр.: 18.

Колосов А.М., Смелянский Р.Л. Исследование подходов к построению информационно-ориентированных сетей // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМиК МГУ, 2016.

В статье представлена концепция построения ICN сети на базе SDN технологии и показано, что такой подход эффективнее по сравнению с подходом построения ICN сети, с помощью технологий традиционных сетей. Эффективность SDN подхода демонстрируется обосновывается на математической модели. Теоретические результаты подкреплены результатами имитационного моделирования. Приведено обоснование выбора критерия эффективности.

Ил.: 2 рис., Библиогр.: 21

Пинаева Н.М., Антоненко В.А. Разработка системы управления виртуальными сетевыми функциями в облачной платформе // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМиК МГУ, 2016.

В работе рассмотрена задача разработки системы управления виртуальными сетевыми функциями в облачной платформе, предложена архитектура модуля, реализующего данную систему управления, и проведено его функциональное тестирование.

Ил.: 4 рис., Библиогр.: 9.

Кочетков П.А., Антоненко В.А. Организация сетевого взаимодействия между компонентами системы NPS // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМиК МГУ, 2016.

Данная статья описывает вариант улучшения системы NPS путем организации сетевого взаимодействия между ее компонентами таким образом, чтобы обеспечить неравномерную загрузку вычислительных ресурсов, а также поддерживать возможность задания пропускной способности и задержки моделируемых каналов. В статье описывается архитектура и принципы работы системы NPS, проводится обзор подходов к решению задачи, а также приводятся детали реализации выбранного подхода.

Ил.: 3 рис., 1 таб. Библиогр.: 10.

Романов А.Р., Антоненко В.А. Разработка системы обеспечения надежного и масштабируемого виртуального сетевого сервиса в облачной среде // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМиК МГУ, 2016.

В статье рассмотрен высокоуровневый стандарт архитектуры NFV платформ ETSI NFV MANO. Проанализированы существующие решения, решающие задачи восстановления сервиса и расширения его инфраструктуры. Предложена архитектура программного модуля, позволяющего управлять виртуальными сетевыми сервиса и обеспечивать их отказоустойчивость и масштабируемость в автоматическом режиме. Проведены эксперименты разработанной архитектуры.

Ил.: 3 рис., Библиогр.: 6.

Балашов В. В., Костенко В. А. Комплексы бортового радиоэлектронного оборудования с архитектурой ИМА // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМиК МГУ, 2016.

В статье рассмотрены особенности построения комплексов бортового радиоэлектронного оборудования с архитектурой интегрированной модульной авионики, задачи, возникающие при построении комплекса, и основные причины, которые могут приводить к снижению эффективности использования аппаратных средств вычислительных и сетевых ресурсов комплекса.

Библиогр.: 10.

Шпилевой В.Д., Герасев А.В. В распределенных вычислительных системах иногда возникает необходимость синхронизации времени на узлах таких систем с некоторым источником сигналов эталонной частоты.

В данной работе рассматривается реализация существующего алгоритма синхронизации времени на машинах с установленной операционной системой Linux с источником pulse per second (PPS) сигналов в качестве источника сигналов эталонной частоты.

Так же представлен ряд модификаций существующего алгоритма, которые были реализованы на языке C в ядре Linux, и численные исследования его эффективности до и после модификации.

Ил.: 10 рис., 3 таб. Библиогр.: 12.

Новоселов А.Д., Балаханов В.А. Построение статико-динамического расписания алгоритмом с обратной связью. // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМиК МГУ, 2016.

В статье приведено описание предложенного и реализованного алгоритма с обратной связью для построения статико-динамического расписания, а также результаты экспериментальной апробации алгоритма.

Ил.: 1 табл., Библиогр.: 3

Запутляев И.А., Волканов Д.Ю. Исследование адаптивного генетического алгоритма для решения задачи оптимизации надежности распределённых вычислительных систем // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМК МГУ, 2016.

В данной работе рассматривается задача оптимизации надежности распределённых вычислительных систем. В задаче требуется найти такую конфигурацию распределённой вычислительной системы, чтобы при заданных ограничениях на общую стоимость надёжность системы была максимальной. В работе приводится описание решения данной задачи с помощью Адаптивного Генетического Алгоритма на основе различных схем адаптации. Предложена реализация этого алгоритма, проведено её экспериментальное исследование, включающее сравнение с классическим Генетическим Алгоритмом и Гибридным Генетическим Алгоритмом со встроенным блоком нечёткой логики.

Ил.: 3 табл., Библиогр.: 11.

Рябов Г.Г. Генетический взгляд на симметрии простых чисел в структуре натуральных // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМК МГУ, 2016.

В статье на уровне, близком к элементарному рассматривается композиция структур n-куба, глобальных k-арных деревьев и множества натуральных N. Свойства этой композиции используются при рассмотрении симметричности простых чисел P в структуре натуральных N на базе разностного таблоида, определенного на P×N.

Ил.: 3 рис., Библиогр.: 9.

Голышев Н.В. Исследование эффективности факторизации больших чисел на графических процессорах // Программные системы и инструменты. Тематический сборник № 16, М.: Изд-во факультета ВМК МГУ, 2016.

Рассматривается этап просеивания алгоритма факторизации общим методом решета числового поля. Исследуется модельный алгоритм просеивания, а также эффективность его реализации на графических процессорах в сравнении с версией для центрального процессора.

Ил.: 3 рис., 3 таб. Библиогр.: 10.

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, 2016. – № 16. – 148 p.

Theese proceedings are dedicated to the memory of L.N. Korolev. They consists of student's works, who have received the recommendation of the Department's scientific seminars, which he created and directed the stretch. The proceedings contains articles devoted to descriptions of software systems tools developed by the authors of publications. They refer to the programming techniques for a super-computer, the analysis of texts in natural languages, mathematical linguistics, tools that provide real-time work management system (RTS), and some issues of theoretical programming. Also features articles and general notes relating to informatics sections closely related to programming.

These papers will be of interest to students, graduate students and professionals in the development of application software systems using new information technologies.

Keywords: software-defined networking, network operating system, computer security attack, centralized controller, information-centric networking, cloud computing, network functions virtualization, simulation&modeling, real-time systems, integrated modular avionics, time synchronization, job scheduling, reliability optimization problem, genetic algorithms, prime numbers, integer factorization.

Научное издание

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 16

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

Напечатано с готового оригинал-макета
Подписано в печать 29.08.2016 г.
Формат 60x90 1/16. Усл.печ.л. 9,25. Тираж 100 экз. Заказ 235.
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.