View metadata, citation and similar papers at core.ac.uk
brought to you by CORE

Моделирование и анализ информационных систем. Т. 26, № 1 (2019), с. 101–121
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 101–121

©Пашков В. Н., 2019

DOI: 10.18255/1818-1015-2019-1-101-121

УДК 517.9

Распределенная отказоустойчивая платформа управления для программно-конфигурируемых сетей

Пашков В. Н.

Поступила в редакцию 10 января 2019
После доработки 15 февраля 2019
Принята к публикации 18 февраля 2019

Аннотация. В рамках исследования рассматривается проблема обеспечения отказоустойчивости распределенной платформы управления для программно-конфигурируемых сетей. Целью исследования является разработка архитектуры и принципов организации отказоустойчивой распределенной платформы управления для ПКС. Отказоустойчивость распределенной платформы управления ПКС достигается за счет резервирования контроллеров, резервирования активных соединений между коммутатором и несколькими контроллерами, резервирования вычислительных ресурсов и использования дополнительных программных инструментов для обнаружения отказов, предотвращения перегрузок и восстановления управления сетью. В работе приводится алгоритм распределения управления коммутаторами между контроллерами платформы управления, выбора резервных контроллеров для каждого коммутатора, что позволяет минимизировать время восстановления в случае одиночных отказов контроллеров. Алгоритм балансировки нагрузки между контроллерами позволяет динамически переконфигурировать платформу управления с минимальным количеством операций передачи управления коммутаторами, чтобы предотвратить перегрузку контроллера. Представлены результаты экспериментального исследования предложенных алгоритмов.

Ключевые слова: программно-конфигурируемые сети, ПКС, распределенная платформа управления, РПУ, отказоустойчивость, балансировка нагрузки, OpenFlow, сетевая архитектура

Для цитирования: Пашков В. Н., "Распределенная отказоустойчивая платформа управления для программно-конфигурируемых сетей", *Моделирование и анализ информационных систем*, **26**:1 (2019), 101–121.

Об авторах:
Пашков Василий Николаевич, программист, orcid.org/0000-0001-5783-4557
Московский государственный университет им. М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: pashkov@lvk.cs.msu.su

Введение

Концепция программно-конфигурируемых сетей (ПКС, SDN) [1–3] предполагает логически централизованное управление сетью, при котором набор функций управления сосредоточен в сетевой операционной системе (или контроллере) и сетевых

приложениях, работающих на выделенном сервере. Контроллер поддерживает и контролирует глобальное состояние сети. Основываясь на глобальном состоянии сети, сетевые приложения контроллера осуществляют логически централизованное управление сетевыми устройствами, политиками и потоками данных в сети.

В статье рассматриваются программно-конфигурируемые сети, в которых взаимодействие между контроллером и сетевыми устройствами осуществляется по протоколу OpenFlow [4]. Архитектура и реализация контроллера не определены спецификацией OpenFlow, поэтому существует множество различных реализаций контроллеров с открытым исходным кодом и проприетарных сетевых операционных систем (NOX [5, 6], Beacon [7, 8], Floodlight [9], OpenMUL [10], RUNOS [11] и другие). Характеристики производительности, масштабируемости и надежности всей ПКС/OpenFlow сети и качество сетевых сервисов для конечных пользователей определяются функциональностью, стабильностью, производительностью контроллера, а также поведением контроллера в критических ситуациях, характеристиками масштабируемости, надежности и безопасности контура управления ПКС.

В то же время контроллер является единственной точкой отказа в ПКС сетях. Сбои сервера и его программные ошибки, перебои с питанием, ошибки контроллера и его сетевых приложений, уязвимости в системе безопасности или несанкционированный доступ к контроллеру могут привести к отказу ПКС контроллера и недоступности сетевых сервисов контроллера для коммутаторов сети. Следовательно, это может привести к частичной или полной потере управления над сетью, отказу сети и недоступности пользовательских сетевых сервисов.

Другим источником проблем для контура управления ПКС является перегрузка контроллера. Это может быть вызвано сбоями сервера и ограничениями его физической производительности, увеличением количества коммутаторов в сети, значительным увеличением количества новых потоков в сети или DDoS-атакой на контроллер. Перегрузка контроллера трафиком управления может привести к увеличению времени отклика контроллера на запросы от коммутаторов и, следовательно, к увеличению времени установления новых потоков для пользовательских сетевых сервисов.

Для внедрения подхода ПКС/OpenFlow в реальных сетях необходимо устранить эти недостатки и обеспечить отказоустойчивость контура управления для коммутаторов в случае отказа контроллера или его перегрузки.

Основным способом достижения отказоустойчивости является резервирование ресурсов в контуре управления. Это может быть реализовано путем распределения управления коммутаторами между несколькими ПКС контроллерами в контуре управления.

Для реализации этой идеи необходимо решить следующие задачи:

1. Как распределить управление коммутаторами между несколькими ПКС контроллерами?

2. Как перераспределить управление в случае отказа одного контроллера в распределенной платформе управления?

3. Как предотвратить перегрузку контроллера в распределенной платформе управления?

В разделе 1 статьи проводится краткий обзор существующих реализаций распределенных контроллеров для ПКС сетей. В разделе 2 предлагается архитектура отказоустойчивой распределенной платформы управления. В разделах 3 и 4 предлагается проактивный алгоритм распределения управления коммутаторами между контроллерами и алгоритм балансировки нагрузки между ними. В разделе 5 приводятся результаты экспериментальных исследований.

1. Обзор распределенных платформ

Приложение HyperFlow (C++, Python, OpenFlow 1.0) [13] для контроллера NOX – это первая попытка реализации распределенного контура управления ПКС с использованием распределенной файловой системы при сохранении логически централизованного управления и поддержки согласованного глобального состояния сети. Каждый коммутатор подключен к контроллеру, до которого задержка является наименьшей, или к контроллеру, указанному сетевым администратором (чтобы минимизировать затраты).

Onix (C++, Python, Java, OpenFlow 1.0) [14] – первая распределенная платформа управления ПКС, изначально предназначенная для виртуализации сетей в центрах обработки данных. Приложения Onix поддерживают согласованное глобальное состояние сети и используют базовые примитивы управления состоянием, предоставляемые платформой. Onix предоставляет общий API для сетевых приложений, позволяя им самостоятельно находить компромиссы между согласованностью, надежностью и масштабируемостью. Для достижения отказоустойчивости платформа Onix устойчива к четырем типам отказов: отказ коммутатора, отказ канала связи между коммутаторами, отказ контролера Onix и отказ канала связи между коммутатором и контроллером Onix (и между контроллерами Onix). Тем не менее, Onix остается закрытой разработкой, и дальнейшее развитие не описано в публикациях.

Контроллер Kandoo (C, C++, Python, OpenFlow 1.0) [15] является иерархически распределенной платформой управления ПКС. Kandoo имеет два уровня контроллеров (и сетевых приложений контроллеров соответственно): корневой (root) контроллер и набор локальных контроллеров. Ключевая особенность контроллера Kandoo – это снижение нагрузки на root-контроллер за счет обработки значительного количества запросов на установление новых потоков от коммутаторов на локальных контроллерах. В свою очередь, root-контроллер представляет собой распределенную платформу управления, такую как Onix. В случае отказа локального контроллера сегмент сети, оставшийся без контроллера, переключается на корневой контроллер.

Реализация распределенных контроллеров ElastiCon, HAC и ONOS использует возможность поддержки активных соединений коммутатора одновременно с несколькими контроллерами, которая была добавлена в протокол OpenFlow, начиная с версии 1.2.

Протокол OpenFlow [12] позволяет коммутатору поддерживать несколько активных защищенных каналов управления с несколькими контроллерами одновременно. Каждый контроллер по отношению к коммутатору может выступить в одной из трех ролей: MASTER, SLAVE или EQUAL. В ролях MASTER и EQUAL контроллер име-

ет полный доступ к коммутатору и может принимать все асинхронные сообщения (например, packet-in сообщения) от коммутатора. В роли SLAVE контроллер имеет доступ только для чтения состояния коммутатора и не может получать от него асинхронные сообщения. Контроллер может изменить свою собственную роль, используя особый тип OpenFlow сообщений для запроса роли (role-request). Эта новая возможность протокола OpenFlow позволяет коммутатору не устанавливать новый защищенный канал в случае сбоя контроллера. Протокол полностью возлагает ответственность за изменение роли на контроллер. Однако для стабильной работы сети для каждого коммутатора должен быть определен свой MASTER контроллер.

ElastiCon (Java, OpenFlow 1.3) [17] – экспериментальная реализация распределенного контроллера ПКС на основе контроллера с открытым исходным кодом Floodlight, который обеспечивает поддержку согласованного глобального состояния сети посредством использования распределенного in-memory хранилища Hazelcast. Ключевой особенностью ElastiCon является динамическая балансировка нагрузки в платформе управления: количество экземпляров контроллера динамически увеличивается или уменьшается в зависимости от условий и количества управляющего трафика. Нагрузка на платформу управления динамически перераспределяется между экземплярами контроллера на основе операции миграции коммутатора.

Контроллер высокой готовности (HAC) (C++, OpenFlow 1.3) [18] является экспериментальной реализацией распределенной платформы управления высокой готовности, основанной на контроллере nox13oflib для корпоративных сетей ПКС/OpenFlow. Для достижения высокой готовности платформа управления на основе контроллера HAC использует стратегию "активный/резервный" для распределения состояния основного контроллера и автоматического переключения на резервный контроллер в случае сбоя основного контроллера. В [18] представлена архитектура платформы управления HAC и реализация прототипа контроллера HAC. В случае отказа основного контроллера резервный контроллер автоматически берет на себя управление сетевой инфраструктурой и управление потоками данных. Эта процедура обеспечивает отказоустойчивость контроллера. Для обнаружения отказа HAC контроллер использует алгоритм Heartbeat с периодической отправкой сообщений основному контроллеру для проверки его работоспособности. Процедура восстановления после отказа основного контроллера может быть реализована двумя способами: либо путем настройки того же сетевого интерфейса, что и на основном контроллере, либо путем изменения роли резервного контроллера на MASTER на каждом сетевом коммутаторе.

Контроллер ONOS (Java, OpenFlow 1.0 и 1.3) [19, 20] – это распределенная сетевая операционная система с открытым исходным кодом для ПКС операторов связи. Высокая производительность платформы управления ONOS достигается за счет разделения сети на сегменты, их распределения между экземплярами контроллера и использования многопоточной обработки сообщений. Для достижения высокой готовности платформа управления ONOS использует избыточные защищенные каналы связи между коммутаторами и контроллерами, глобальное распределение состояния сети между контроллерами платформы, распределение ролей контроллера для каждого коммутатора и процедуру изменения роли контроллера в случае отказа основного контроллера. Однако ONOS не поддерживает процедуру балансировки нагрузки между экземплярами контроллера.

2. Предлагаемая архитектура отказоустойчивой распределенной платформы управления

Программно-конфигурируемая сеть с отказоустойчивым распределенным контуром управления должна включать в себя четыре основных уровня (см. Рисунок 1):

1. Контур данных.
2. Инфраструктура подключения OpenFlow.
3. Распределенная платформа управления.
4. Межконтроллерная коммуникационная инфраструктура.

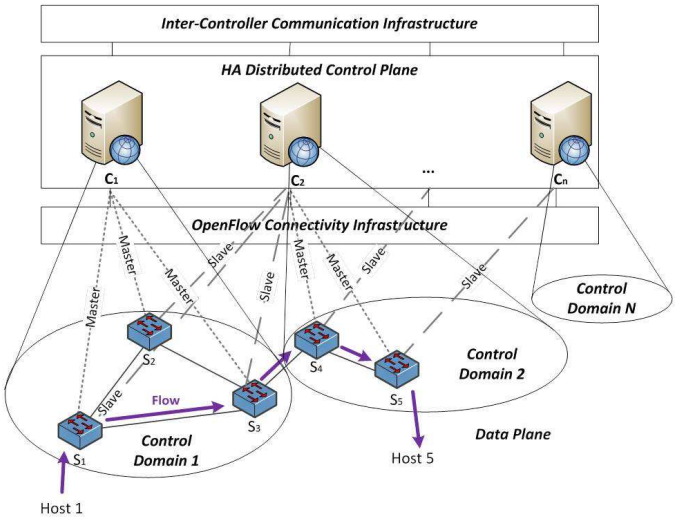

Рис. 1. Модель отказоустойчивой распределенной платформы управления ПКС

Fig. 1. The model of high-availability distributed control plane for software-defined networks

Контур данных (Data Plane) состоит из объектов управления: сетевых устройств (коммутаторов) и потоков данных. Контур данных предназначен для предоставления сетевых сервисов конечным пользователям. Мы предполагаем, что коммутаторы работают в соответствии со спецификацией коммутатора OpenFlow и поддерживают протокол OpenFlow (не ниже версии 1.2).

Инфраструктура подключения OpenFlow предназначена для обеспечения взаимодействия между контуром данных и контуром управления на основе протокола OpenFlow. Инфраструктура связи обеспечивает управление сетевыми устройствами и передачей трафика; управляющий трафик физически отделен от трафика данных. Также этот уровень включает в себя совокупность логических защищенных каналов связи "контроллер-коммутатор" [12].

Распределенная платформа управления (РПУ) включает физически распределенную и логически централизованную сетевую операционную систему с набором сетевых приложений для управления коммутаторами и потоками данных.

Распределенная сетевая операционная система (РСОС) состоит из набора контроллеров. РСОС поддерживает актуальное глобальное состояние сети, которое содержит состояние коммутаторов, каналов, портов, таблиц потоков и другие данные.

Инфраструктура связи между контроллерами включает в себя выделенную сеть для обеспечения связи между контроллерами распределенной платформы управления и протокол взаимодействия контроллеров. Он предназначен для мониторинга состояния контроллеров РПУ, координации контроллеров РПУ, обмена сообщениями между контроллерами и распределения глобального состояния сети между контроллерами РПУ.

На основе этой архитектуры РПУ можно выделить домены и домены управления для каждого контроллера. Домен включает в себя один экземпляр контроллера и набор коммутаторов, с которыми экземпляр контроллера поддерживает активное соединение. Домен управления включает в себя набор коммутаторов и один экземпляр контроллера, который является мастером для этих коммутаторов. Каждый коммутатор должен быть включен в домен управления, но только в один. Таким образом, можно определить Control Domain View (CDV) как набор всех доменов управления в РПУ. И для каждого коммутатора может быть определен групповой контроллер как набор экземпляров контроллера РПУ, которые поддерживают активное соединение (защищенный канал) с коммутатором и имеют согласованную информацию о текущем состоянии коммутатора. Таким образом, любой из экземпляров группового контроллера может быть определен как потенциально основной контроллер для этого коммутатора.

Потоки данных в сети можно классифицировать следующим образом: внутридоменные, транзитные, входящие и исходящие потоки данных. Внутридоменный поток данных не выходит за пределы сетевого сегмента и не требует участия других экземпляров контроллера для маршрутизации и передачи данных. Транзитные, входящие и исходящие потоки требуют координации между контроллерами и взаимодействия между доменами. Таким образом, загрузка РПУ и отдельных экземпляров контроллера зависит от распределения коммутаторов между доменами управления.

Для поддержки отказоустойчивости каждый экземпляр контроллера РПУ должен включать следующие дополнительные программные компоненты:

1. сервис обнаружения отказов контроллера,
2. сервис обнаружения перегрузки контроллера,
3. сервис восстановления управления,
4. сервис координации контроллеров,
5. сервис синхронизации контроллеров.

Сервис обнаружения отказов контроллера используется для мониторинга работоспособности контроллеров РПУ на основе метода Heartbeat.

Сервис обнаружения перегрузки контроллера включает в себя возможность отслеживать загрузку экземпляров контроллера (количество коммутаторов и потоков в каждом домене управления, частоту пакетов и задержку установления

новых потоков для каждого экземпляра контроллера, использование центрального процессора и памяти каждого сервера). Каждый индикатор загрузки экземпляра контроллера имеет фиксированное или плавающее предельное значение. Когда предел превышен, экземпляр контроллера регистрирует факт потенциальной перегрузки контроллера.

Сервис восстановления управления включает в себя набор алгоритмов и сценариев поведения экземпляра контроллера в случае реализации угроз (отказ одного экземпляра контроллера РПУ, перегрузка контроллера, потеря соединения между коммутатором и его основным контроллером).

Сервис координации контроллеров использует распределенный алгоритм консенсуса на основе Paxos [23] для координации событий смены ролей между контроллерами, событий обнаружения и восстановления и других событий в распределенной платформе управления.

Сервис синхронизации контроллеров использует распределенный алгоритм для синхронизации хранилищ данных с информацией о глобальном состоянии сети между контроллерами РПУ.

3. Распределение управления коммутаторами между контроллерами с учетом резервирования

В данной работе рассматривается ПКС/OpenFlow сеть с распределенной платформой управления, как показано на Рис. 1. $G = (S, E)$ – это граф сети. Контур данных включает в себя множество коммутаторов $S = \{s_1, s_2, ..., s_m\}$ из m коммутаторов. Распределенная платформа управления включает в себя множество $C = \{c_1, c_2, ..., c_n\}$ из n ПКС/OpenFlow контроллеров, запущенных на одинаковых серверах.

$E = \{e_{ij} = (s_i, s_j) | s_i, s_j \in S\}$ – множество каналов связи между коммутаторами.

$E_{sc} = \{e_{ij} = (s_i, c_j) | s_i \in S, c_j \in C\}$ – множество всех возможных логических защищенных каналов связи между коммутаторами и контроллерами. Для каждого защищенного канала связи может быть определена одна из четырех ролей $R = \{Master, Equal, Slave, None\}$ и отношение $role(c_i, s_j) \in R$. $None$ означает, что соединение между контроллером и коммутатором отсутствует.

Для обеспечения стабильного функционирования сети необходимо, чтобы для каждого коммутатора был определен единственный контроллер в роли $Master$. Для каждого коммутатора s_i существует, и только один, контроллер $c_j : role(c_j, s_i) = Master$.

Следовательно, для каждого контроллера может быть определен сегмент контура данных, который включает в себя множество коммутаторов, для которых он является $Master$ контроллером. Пусть $group(c_i) = \{s_j \in S \mid role(c_i, s_j) = Master\}$ – множество коммутаторов, для которых c_i – это $Master$ контроллер. И для каждого коммутатора s_i можно определить групповой контроллер $GC(s_i) = \{c_j \in C \mid role(c_j, s_i) \neq None\}$.

Для обеспечения отказоустойчивости для каждого коммутатора s_i должно быть $|GC(s_i)| \geq 2$.

$S' = \{S'_1 \dots S'_{|C|}\}$ — множество групп коммутаторов $S'_j \subseteq S$, где контроллер

$c_j \in C$ является $Master$ контроллером для каждого коммутатора $s_i \in S'_j$. Таким образом, $S'_j = \{s_i \in S \mid role(c_j, s_i) = Master\}$.

Свойства корректности организации управления в ПКС с распределенным контроллером можно сформулировать следующим образом:

Свойство безопасности: В любой момент времени для каждого коммутатора $s_i \in S$ существует не более одного $Master$ контроллера c_j: $role(c_j, s_i) = Master$.

Свойство живучести: В конечном итоге для каждого коммутатора $s_i \in S$ некоторый активный контроллер $c_j : role(c_j, s_i) \neq None$ должен стать $Master$ контроллером для него.

Пусть физическое размещение контроллеров РПУ ПКС статически определено в сети. Коммутаторы могут быть непосредственно соединены с контроллерами с использованием out-of-band каналов управления, остальные коммутаторы – посредством in-band каналов управления. Каждый контроллер может быть в состоянии активный или неактивный. Пусть каждый коммутатор поддерживает логическое соединение (защищенный канал) с каждым активным контроллером РПУ. В соответствии со спецификацией OpenFlow 1.3 для каждого коммутатора должен быть определен по крайней мере один $Master$ контроллер. Предполагается, что другие контроллеры для этого коммутатора находятся в роли $Slave$. И для каждого коммутатора в сети необходимо определить резервный контроллер – один из контроллеров в роли $Slave$, который перехватит управление сегментом сети в случае отказа $Master$ контроллера.

Таким образом, можно сформулировать задачу распределения управления коммутаторами между контроллерами для РПУ ПКС: для данной конфигурации активных контроллеров необходимо определить основной ($Master$) и резервный контроллер для каждого коммутатора.

3.1. Алгоритм выбора основного контроллера

Введем следующие обозначения:

- $d^0_{i,j}$ — задержка канала связи e_{ij} между коммутаторами s_i и s_j,
- d^j_i — задержка кратчайшего пути между коммутаторами s_i и s_j как сумма задержек каналов связи, входящих в кратчайший путь,
- N_{max} — максимальное количество коммутаторов, которыми может управлять один контроллер. Этот параметр одинаков для всех контроллеров,
- $N(c_j) = |S'_j|$ - количество коммутаторов, для которых контроллер c_j является $Master$ контроллером, для каждого контроллера c_j: $N(c_j) \leq N_{max}$.

Метрики задержки для основного контроллера могут быть определены следующим образом:

1. Средняя задержка между контроллерами и коммутаторами в сети:

$$L_{avg}(S') = \frac{1}{|S|} \sum_{s_i \in S} d^j_i, \quad c_j \in C, \quad s_i \in S'_j. \tag{1}$$

2. Задержка в худшем случае в сети между контроллерами и коммутаторами:

$$L_{wc}(S') = \max_{s_i \in S} d_i^j, \quad c_j \in C, \quad s_i \in S'_j. \tag{2}$$

Пусть S' – текущее размещение контроллеров в сети.

Можно сформулировать задачу определения основных контроллеров РПУ для коммутаторов следующим образом:

$$\begin{cases} \min L_{avg}(S') \quad or \quad \min L_{wc}(S') \\ N(c_j) \leq N_{max} \quad \forall j = \overline{1, |C|} \end{cases} \tag{3}$$

Решением этой задачи является распределение управления коммутаторами между контроллерами РПУ ПКС: $S' = \{S'_1 \dots S'_{|C|}\}$ — множество групп коммутаторов $S'_j \subseteq S$, где контроллер $c_j \in C$ является $Master$ контроллером для каждого коммутатора $s_i \in S'_j$. То есть $S'_j = \{s_i \in S \mid role(c_j, s_i) = Master\}$.

Для решения задачи выбираются потенциальные узлы сети, в которых могут быть размещены активные контроллеры РПУ ПКС, на основе использования алгоритма поиска двусвязных компонентов в графе. Для решения проблемы определения основных контроллеров для коммутаторов используются методы k-средних или k-medians в зависимости от выбранной метрики минимизации задержек.

3.2. Алгоритм выбора резервного контроллера

Для обеспечения отказоустойчивости управления в случае отказа контроллера или отказа соединения контроллер–коммутатор (защищенного канала связи) необходимо определить контроллер РПУ ПКС, который будет резервным контроллером для данного коммутатора.

Требования к резервным контроллерам в сети:

- Для каждого коммутатора s_i расположение резервного контроллера c_j отличается от основного контроллера c_k, то есть $j \neq k$.
- Для каждого контроллера c_j количество управляемых коммутаторов в сети не более N_{max}: $N(c_j) \leq N_{max}$.
- Платформа управления поддерживает механизм распределения глобального состояния сети между контроллерами РПУ.

Используя следующие обозначения, можно сформулировать задачу выбора резервного контроллера:

- $x_i^j \in \{0, 1\}$ — индикатор, $x_i^j = 1$, если контроллер $c_j \in C$ назначен резервным контроллером для коммутатора $s_i \in S$, и $x_i^j = 0$ — в противном случае;
- d_i^j — задержка кратчайшего пути между контроллером c_i и коммутатором s_j (из D^N – матрица кратчайших путей);
- $k_i \in C$ — основной контроллер для коммутатора s_i (k_i в $Master$ роли для коммутатора);

- S_i — набор коммутаторов, для которых контроллер c_i является основным контроллером;
- $M_i = |S_i| - N_{max}$ — максимальное количество коммутаторов, для которых контроллер c_i может быть назначен как резервный контроллер.

Выбор целевой функции зависит от выбранного критерия оптимизации:

1. Минимизация средней задержки между контроллером и коммутатором (L_{avg}):
$$F = \sum_{i=\overline{1,|S|}} \sum_{j=\overline{1,|C|}} d_i^j x_i^j \ . \tag{4}$$
2. Минимизация задержки в худшем случае между контроллером и коммутатором (L_{wc}):
$$F = \max_{i=\overline{1,|S|}} \sum_{j=\overline{1,|C|}} d_i^j x_i^j \ . \tag{5}$$

Формулировка проблемы выбора резервных контроллеров

Можно сформулировать следующую **логическую задачу линейного программирования**, чтобы определить резервные контроллеры для каждого коммутатора в сети:
$$\begin{cases} \min F & \\ x_i^j = 0 & \forall i,j:\ k_i = c_j \\ \sum\limits_{j=\overline{1,|C|}} x_i^j = 1 & \forall i = \overline{1,|S|} \\ \sum\limits_{s_i \in S_p} x_i^j \le M_j & \forall j, p = \overline{1,|C|} \end{cases} \tag{6}$$

Можно оценить минимальное количество контроллеров для поддержки резервных контроллеров для каждого коммутатора в сети. Для сети из $|S|$ коммутаторов необходимо следующее количество контроллеров, чтобы обеспечить отказоустойчивость контура управления к одиночному отказу контроллера:
$$|C| \ge \left\lceil \frac{|S|}{N_{max}} \right\rceil + 1 \ . \tag{7}$$

В этом случае для любых начальных распределений коммутаторов между экземплярами контроллера (таких, что в любой группе контроллеров не более N_{max} коммутаторов), можно назначить резервный контроллер для каждого коммутатора. А в случае сбоя любого экземпляра контроллера в сети количество коммутаторов, управляемых одним контроллером, не превышает N_{max}.

В общем случае для устойчивости к сбоям N контроллеров в сети требуется следующее количество контроллеров:
$$|C| \ge \left\lceil \frac{|S|}{N_{max}} \right\rceil + N \ . \tag{8}$$

Поскольку проблема выбора резервного контроллера была сведена к булевой задаче линейного программирования, был реализован алгоритм, основанный на методе Балаша [22]. Алгоритм позволяет найти точное решение со значительным сокращением пространства решения. Алгоритм обеспечивает сходимость с использованием конечного числа итераций.

4. Балансировка нагрузки между контроллерами

Загрузка экземпляра контроллера создается путем обработки входящих сообщений. Для установления нового потока данных на контроллере генерируется набор правил для коммутаторов. Таким образом, загрузка контроллера – это количество входящих сообщений Packet-In в секунду, а производительность или пропускная способность контроллера – это максимальное количество Packet-In сообщений, которые он может обработать в секунду.

Во время работы сети характер, динамика, объемы трафика и количество потоков данных динамически изменяются, и поэтому также изменяется нагрузка на контур управления, которая генерируется коммутаторами. Для достижения отказоустойчивости контура управления необходимы специальные механизмы для предотвращения перегрузки любого контроллера РПУ. А в случае низкой загрузки контура управления количество активных контроллеров должно быть минимальным.

Эти проблемы в контуре управления могут быть решены путем динамического перераспределения управления коммутаторами между контроллерами. Для реализации перераспределения управления коммутатором между контроллерами может использоваться алгоритм передачи управления коммутатором из [17].

Чтобы решить проблему распределения нагрузки между экземплярами контроллера РПУ, введем следующие допущения и ограничения:

1. Каждый коммутатор подключен к каждому экземпляру контроллера.
2. Все экземпляры контроллера имеют одинаковую максимальную производительность.
3. Для каждого коммутатора в каждый момент времени должен быть определен только один мастер.
4. Экземпляры контроллера используют проактивный режим установки правил, то есть контроллер устанавливает правила в коммутаторы, для которых он определен как мастер.

Таким образом, можно декомпозировать задачу на следующие подзадачи:

1. Подзадача группировки коммутаторов: необходимо разделить все коммутаторы на группы в соответствии с их нагрузкой, чтобы любая групповая нагрузка не превышала производительность контроллера.
2. Подзадача распределения групп между экземплярами контроллеров: необходимо распределить группы коммутаторов между контроллерами с минимальным количеством передач управления коммутаторами.

4.1. Алгоритм группировки коммутаторов

Пусть в контуре управления p_i является максимально допустимой пропускной способностью/производительностью контроллера c_i (количество входящих сообщений в секунду), а $load(c_i)$ – средняя текущая нагрузка операций установки потоков для контроллера $c_i \in C$.

Поскольку используется набор идентичных экземпляров контроллера, то $p_i = p$ для всех $c_i \in C$. Значение p задано.

Пусть задана некоторая статистическая картина потоков в контуре данных: $inf(s_j)$ – среднее число новых потоков в секунду в коммутатор s_j и $f_{ij} = f(s_i, s_j)$ – это среднее число потоков данных в секунду от коммутатора s_i к коммутатору s_j.

Проблема группировки коммутаторов: распределить коммутаторы по группам так, чтобы для каждого контроллера c_i: $load(c_i) \leq p$. И требуется достичь минимального количества активных контроллеров в контуре управления. Математическая формулировка **подзадачи группировки коммутаторов** следующая:

$$\min \sum_{i=1}^{n} y_i$$

$$\text{при условии} \quad load(c_i) \leq p,\ c_i \in C,$$

$$\sum_{i=1}^{n} x_{ij} = 1, j \in \{1, \ldots, n\}$$

где

$$y_i = \begin{cases} 1, \text{если контроллер } c_i \text{ активен,} \\ 0, \text{в противном случае.} \end{cases}$$

$$x_{ij} = \begin{cases} 1, \text{если } c_i - Master \text{ для коммутатора } s_j, \\ 0, \text{в противном случае.} \end{cases}$$

Эта задача является NP-трудной. Для решения этой задачи предлагается жадный алгоритм для распределения коммутаторов по минимальному количеству групп и соответственно минимальному числу контроллеров для них.

Алгоритм группировки коммутаторов включает в себя следующие шаги:

1. Вычислить общее количество потоков данных, проходящих через каждый канал связи $e_{ij} \in E$ в контуре данных:

$$l_{ij} = l(s_i, s_j) = f_{ij} + f_{ji}.$$

2. Вычислить общее количество входящих потоков данных для каждого коммутатора $s_i \in S$:

$$f(s_i) = \sum_{s_k \in S} f(s_k, s_i) + inf(s_i).$$

3. Отсортировать каналы связи e_{ij} из E по убыванию общего количества l_{ij} потоков данных.

4. Проинициализировать новый граф $G' = (S, E')$, где $E' = \emptyset$. G' является несвязным графом с $|S|$ компонентами связности.

5. Для группы коммутаторов, пока набор каналов связи E не пустой, необходимо выполнять следующие действия:

   (a) Выбрать линк $e_{ij} \in E$ с максимальным l_{ij},

(b) Удалить линк e_{ij} из E: $E = E \setminus \{e_{ij}\}$,

(c) Добавить линк e_{ij} в E: $E' = E' \cup \{e_{ij}\}$,

(d) Рассчитать нагрузку для каждой группы коммутаторов, которая соответствует компоненту связности графа G', используя следующую формулу:

$$p_j = f_j + \sum_{k=1}^{n} c_{kj}(1 - t_{jk}),$$

$$\text{где } t_{jk} = \begin{cases} 1, \text{если } s_j \text{ и } s_k \text{ в одной группе,} \\ 0, \text{в противном случае.} \end{cases}$$

(e) Добавить коммутатор в группу.

Таким образом, получаем граф G' с разбиением сети на сегменты, распределение коммутаторов в группы, для которых требуется минимальное количество контроллеров РПУ ПКС.

4.2. Алгоритм распределения групп коммутаторов между контроллерами

Чтобы сократить время балансировки нагрузки в контуре управления, необходимо минимизировать количество операций переключения управления коммутаторами между текущим распределением Мастер-контроллеров и требуемым распределением Мастер-контроллеров после перебалансировки. Предполагается, что количество активных контроллеров может быть изменено после процедуры балансировки нагрузки.

Пусть контур управления включает в себя m активных контроллеров, которые управляют контуром данных из n коммутаторов. Пусть матрица $\tilde{X} = [\tilde{x}_{ij}]_{m \times n}$ – это текущее представление распределения Мастер-контроллеров в сети, где $\tilde{x}_{ij} = 1$, если контроллер c_i управляет коммутатором s_j, $\tilde{x}_{ij} = 0$ – в противном случае.

Пусть X – матрица групп коммутаторов, полученная из алгоритма группировки коммутаторов. $x = \{x_1, \ldots, x_n\}$ – группа коммутаторов, где $x_j = 1$, если коммутатор s_j включен в группу и $x_j = 0$ в противном случае. И необходимо предложить набор операций передачи управления коммутаторами для преобразования матрицы $\tilde{X}$ в матрицу X.

Если $|X| \neq m$, то

- Если $|X| < m$, то добавляется $(m - |X|)$ нулевых векторов(строк) в X,
- Если $|X| > m$, то добавляется $(|X| - m)$ нулевых векторов(строк) в $\tilde{X}$.

Чтобы решить эту задачу, необходимо пронумеровать векторы X, чтобы минимизировать следующую функцию:

$$f = \sum_{i=1}^{n} d(x_i, \tilde{x}_i),$$

где $d(x_i, \tilde{x}_i)$ – расстояние Хэмминга между x_i и $\tilde{x}_i$.

Для решения этой задачи предлагается **алгоритм распределения групп между контроллерами** в контуре управления. Он включает в себя следующие шаги:

1. Для каждого коммутатора s_i просмотрим все позиции от 1 до n, чтобы найти в матрице $\tilde{X}$ и в матрице X позицию i, в которой стоит единица. Вычислим расстояние Хэмминга между этими строками.

2. Если добавляются дополнительные строки в $\tilde{X}$, вычислить расстояние Хэмминга между каждым вектором X и нулевым вектором.

3. Сортировать эти пары в порядке возрастания расстояния Хэмминга.

После выполнения этих шагов для каждого x из X найдем $\tilde{x}_i$ из $\tilde{X}$. Это будет означать, что данному элементу x следует присвоить номер i. И определяются контроллеры для новых групп коммутаторов после перебалансировки.

5. Экспериментальные исследования

5.1. Оценка алгоритма распределения коммутаторов между основными и резервными контроллерами

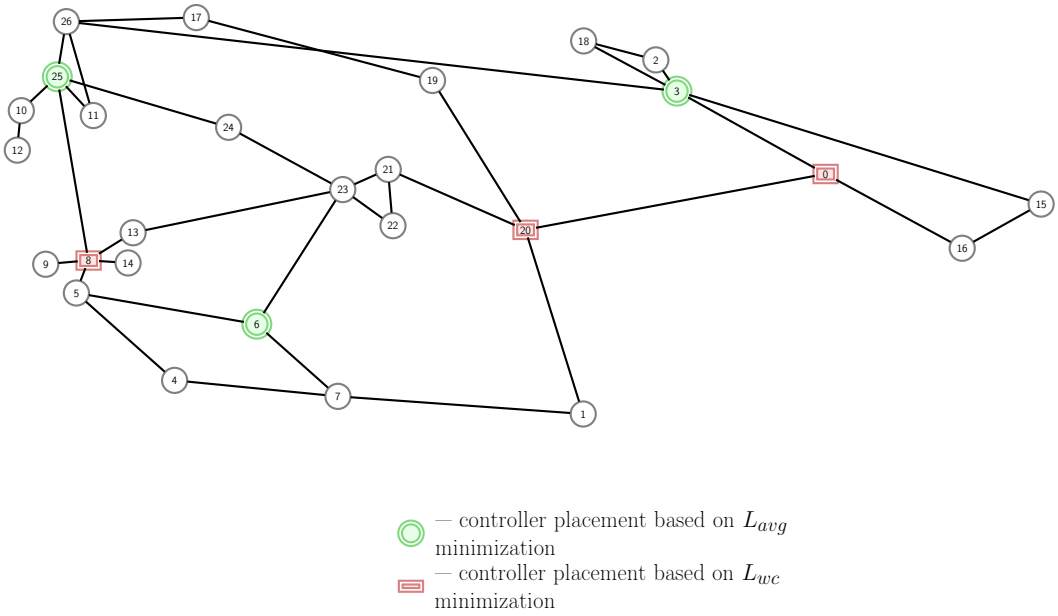

Рис. 2. Размещение контроллеров отказоустойчивой РПУ ПКС в узлах сети Integra Telecom по критериям средней задержки L_{avg} и наихудшей задержки L_{wc}

Fig. 2. Controllers' placement of high-availability distributed control platform for Integra Telecom network by criteria L_{avg} and L_{wc}

Алгоритмы выбора основного и резервного контроллеров реализованы на C ++ с использованием QT5. Для экспериментов используются топологии сети из библиотеки Интернет-топологий TopologyZoo [24] реальных сетей провайдеров.

На рисунке 2 можно увидеть топологию реальной сети Integra Telecom (август 2010 г.), которая состоит из 27 узлов (коммутаторов). Все узлы пронумерованы от 0 до 26. Предполагаем, что контроллеры РПУ ПКС могут быть расположены в любом узле сети. Предполагается, что сеть Integra использует SDN/OpenFlow для управления, а размещение основных контроллеров определяется для случаев минимизации метрик средней задержки между коммутатором и контроллером и задержки в худшем случае.

Результаты алгоритма выбора резервных контроллеров для сети Integra SDN показаны в Таблице 1 и Таблице 2 для средней задержки (L_{avg}) и наихудшей задержки (L_{wc}). В первом столбце можно увидеть пару контроллеров (Master, Slave), а во втором столбце – набор коммутаторов под управлением этих контроллеров.

Таблица 1. Распределение ролей контроллеров отказоустойчивой РПУ ПКС для управления коммутаторами в сети Integra Telecom для случая L_{avg}

Table 1. Assignment of controllers' roles (Master/Slave) for switches in Integra Telecom network in case of L_{avg}

Контроллеры (Master, Slave)	**Коммутаторы**
(3, 6)	0, 15, 16
(3, 25)	2, 3, 18
(6, 3)	1
(6, 25)	4, 6, 7, 20 – 23
(25, 3)	17
(25, 6)	5, 8 – 14, 19, 24, 25, 26

Таблица 2. Распределение ролей контроллеров отказоустойчивой РПУ ПКС для управления коммутаторами в сети Integra Telecom для случая L_{wc}

Table 2. Assignment of controllers' roles (Master/Slave) for switches in Integra Telecom network in case of L_{wc}

Контроллеры (Master, Slave)	**Коммутаторы**
(0, 8)	
(0, 20)	0, 2, 3, 15, 16, 18
(8, 0)	6
(8, 20)	4, 5, 8 – 14, 17, 25, 26
(20, 0)	1, 19 – 24
(20, 8)	

Увеличение задержки в среднем (L_{avg}) и задержки в худшем случае (L_{wc}) при отказе контроллера приведено для различных сетей в Таблице 3 и Таблице 4 (после

распределения происходит переключение между контроллерами с использованием алгоритма выбора резервных контроллеров).

Таблица 3. Увеличение максимальной задержки L_{avg}
в случае отказа контроллера для различных реальных топологий сетей

Table 3. Increasing the average latency L_{avg}
in case of controller failure for different real world network topologies

Название сети	**|S|**	**|C|**	**min, %**	**max, %**	**mid, %**
HiberniaCanada	10	2	60,08	61,82	60,95
Abilene	11	2	117,02	218,67	167,85
Compuserve	11	2	43,97	188,98	116,48
Navigata	13	2	103,28	193,78	148,53
Nsfnet	13	2	65,53	189,65	127,59
Claranet	15	2	77,48	186,82	132,15
Garr199901	16	2	88,72	107,90	98,31
Peer1	16	2	45,95	120,17	83,06
Ernet	16	2	764,00	1427,25	1095,63
Goodnet	17	2	102,56	212,38	157,47
Arpanet19719	18	2	4,48	40,82	22,65
Ibm	18	2	19,38	20,14	19,76
Internetmci	19	2	31,52	51,45	41,49
GtsRomania	19	2	69,68	577,23	323,45
Quest	20	2	147,99	526,47	337,23
BtEurope	22	3	11,50	34,46	20,54
York	23	3	45,11	98,10	78,85
Funet	24	3	147,71	233,42	187,62
Psinet	24	3	1,89	21,11	11,45
Agis	25	3	12,79	31,26	24,58
Integra	27	3	45,17	91,87	69,95
Biznet	28	3	102,10	745,80	519,16
Darkstrand	28	3	22,05	39,01	28,96
Digex	31	3	27,97	84,29	59,05
Bics	33	3	22,06	73,46	39,63
BtNorthAmerica	33	3	16,38	57,86	36,16
Grnet	34	3	0,18	14,59	5,68
NetworkUsa	35	3	6,65	24,51	13,67
Geant2012	37	3	3,73	7,77	6,13
Renater2010	37	3	25,95	191,51	127,11
Cesnet200706	38	3	36,10	126,26	66,29
Chinanet	38	3	16,67	21,41	19,01
Garr200912	42	4	4,73	14,76	10,53
Garr201101	44	4	8,00	41,67	20,16

Таблица 4. Увеличение максимальной задержки L_{wc}
в случае отказа контроллера для различных реальных топологий сетей

Table 4. Increasing the average latency L_{wc}
in case of controller failure for different real world network topologies

Название сети	**|S|**	**|C|**	**min, %**	**max, %**	**mid, %**
HiberniaCanada	10	2	0	30,78	15,39
Abilene	11	2	156,70	204,02	180,36
Compuserve	11	2	86,38	105,66	96,02
Navigata	13	2	132,4	153,8	143,1
Nsfnet	13	2	70,9	105,76	88,33
Claranet	15	2	83,49	113,98	98,74
Garr199901	16	2	71,99	87,33	79,66
Peer1	16	2	10,72	31,37	21,04
Ernet	16	2	335,68	435,68	385,68
Goodnet	17	2	116,16	160,11	138,13
Arpanet19719	18	2	0	0,04	0,02
Ibm	18	2	0	33,474	16,74
Internetmci	19	2	0	33,58	16,792
GtsRomania	19	2	93,57	97,796	95,68
Quest	20	2	304,38	318,8	311,59
BtEurope	22	3	0	15,22	5,08
York	23	3	97,83	151,43	115,7
Funet	24	3	364,52	393,05	376,67
Psinet	24	3	0	10,42	4,67
Agis	25	3	0	9,72	3,24
Integra	27	3	109,99	121,23	114,77
Biznet	28	3	537,52	565,87	550,04
Darkstrand	28	3	0	21,55	11,96
Digex	31	3	15,38	115,38	76,07
BtNorthAmerica	33	3	51,37	151,37	85,31
Bics	33	3	0	57,21	20,97
Grnet	34	3	0	5,75	1,92
NetworkUsa	35	3	0	0,24	0,08
Geant2012	37	3	0	2,28	0,76
Cesnet200706	38	3	4,93	99,03	67,33
Uunet	42	4	0	19,46	8
Garr201101	44	4	0	6,1	1,53
Surfnet	50	4	0	55,9	27,73

В таблицах приведен набор различных сетей, количество коммутаторов и контроллеров в каждой сети, а также минимальное, среднее и максимальное время восстановления в процентах после отказа одного контроллера в сети.

Как видно из этих таблиц, для некоторых сетей средняя задержка между коммутатором и его основным контроллером после сбоя одного контроллера и перераспределения управления в сети увеличивается незначительно. Но для некоторых сетей средняя задержка увеличивается во много раз. Таким образом, для таких сетей общая производительность контура управления может быть достигнута с учетом резерва в случае отказа одного контроллера, но для сохранения приемлемого отклика (времени ответа контроллера) необходимо расширить контур управления, добавив дополнительные контроллеры.

Предложенный алгоритм выбора резервных контроллеров может использоваться не только в режиме реального времени для контура управления для генерации сценариев восстановления в случае отказа одного контроллера, но также и на этапе проектирования контура распределенного управления для конкретной ПКС/OpenFlow сети для выбора разумного количества контроллеров.

5.2. Оценка алгоритма балансировки нагрузки

Предварительные результаты алгоритма группировки коммутаторов показаны на рисунке 3.

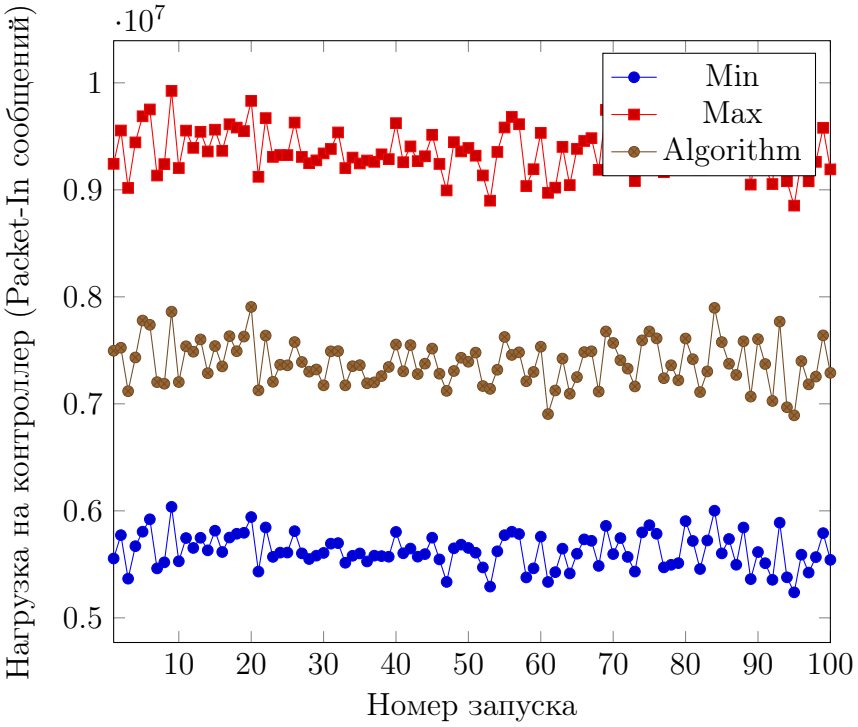

Рис. 3. Общая нагрузка на платформу управления в зависимости от номера запуска

Fig. 3. Total control plane loading depending on the launch number

Исходными данными для алгоритма является топология сети [25], которая включает 141 коммутатор и 748 каналов связи между ними. Для генерации нагрузки для каждого коммутатора в сети использовали равномерное распределение. Алгоритм

был выполнен 100 раз. Для каждого запуска рассчитывается минимальная нагрузка в сети в случае одного контроллера в контуре управления, а максимальная нагрузка в случае одного контроллера в каждом узле сети. И можно наблюдать результат управления загрузкой трафика с помощью алгоритма группировки коммутаторов для балансировки нагрузки между минимальными и максимальными значениями. На рисунке 4 приведено количество сегментов в сети для этого случая.

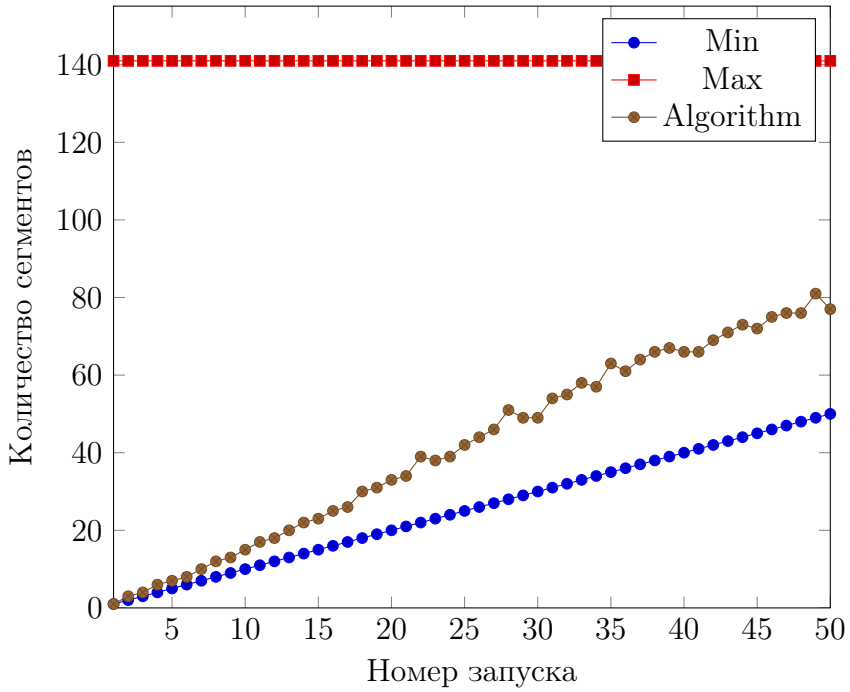

Рис. 4. Количество сегментов в сети в зависимости от номера запуска

Fig. 4. Quantity of segments in the network depending on the launch number

6. Заключение

В этой статье была предложена архитектура отказоустойчивой распределенной платформы управления для глобальных ПКС/OpenFlow сетей. Для достижения отказоустойчивости используется резервирование контроллеров платформы управления, резервирование активных соединений коммутаторов с контроллерами. В работе предложены механизмы для обнаружения и восстановления управления РПУ ПКС. В случае одиночного отказа контроллера РПУ или перегрузки контроллера управление сетью восстанавливается за счет перераспределения управления коммутаторами между оставшимися функционирующими контроллерами РПУ. Предложен проактивный алгоритм выбора резервных контроллеров для каждого коммутатора, который позволяет минимизировать время восстановления в случае сбоя одного контроллера. Предложен алгоритм балансировки нагрузки контроллеров РПУ, чтобы предотвратить перегрузки контроллеров платформы. Эти алгоритмы являются частью сервиса восстановления для каждого РПУ ПКС.

Список литературы / References

[1] McKeown N., et al., "Openflow: Enabling innovation in campus networks", *ACM Computer Communication Review*, **38**:2, (2008), 69–74.

[2] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks", *ONF White Paper*, 2012.

[3] Смелянский Р. Л., "Программно-конфигурируемые сети", *Открытые системы. СУБД*, **9** (2012), 15–26; [Smeliansky R.L., "Software Defined Network", *Open Systems. DBMS*, **9** (2012), 15–26, (in Russian).]

[4] Open Networking Foundation, "OpenFlow Switch Specification, Version 1.0.0 (Wire Protocol 0x01)", *ONF*, 2009.

[5] Gude N., et al., "NOX: towards an operating system for networks", *SIGCOMM Computer Communication Review*, **38**:3 (2008), 105–110.

[6] *NOX OpenFlow Controller*, http://http://www.noxrepo.org/.

[7] Erickson D., "The Beacon OpenFlow controller", *Proceedings HotSDN*, August, 2013.

[8] "Beacon OpenFlow Controller", https://openflow.stanford.edu/display/Beacon.

[9] "Floodlight OpenFlow Controller", http://floodlight.openflowhub.org.

[10] "OpenMul OpenFlow/SDN Controller", http://www.openmul.org/.

[11] "RUNOS OpenFlow Controller", https://github.com/ARCCN/runos.

[12] Open Networking Foundation, "OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04)", *ONF*, 2012.

[13] Tootoocian A., Ganjali Y., "HyperFlow: A distribute control plane for OpenFlow", *Proceedings of the 2010 INM conference/WREN workshop*, 2010, 3.

[14] Koponen T., et al., "Onix: A distributed control platform for large-scale production networks", *OSDI'10*, USENIX, 2010.

[15] Yeganeh S.H., Kandoo Y.G., "A Framework for Efficient and Scalable Offloading of Control Applications", *Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12*, ACM, New York, NY, USA, 2012, 19–24.

[16] Phemius K., Bouet M., Leguay J., "Disco: Distributed multi-domain sdn controllers", *Network Operations and Management Symposium (NOMS)*, IEEE, 2014, 1–4.

[17] Dixit A., et al., "Towards an Elastic Distributed SDN Controller", *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN'13*, ACM, New York, NY, USA, 2013, 7–12.

[18] Pashkov V., Shalimov A., Smeliansky R., "Controller Failover for Enterprise SDN", *Proceedings of the Modern Networking Technologies (MoNeTec'2014)*, IEEE, 2014, 27–29.

[19] Lantz B., et al., "ONOS: Towards an Open, Distributed SDN OS", *ACM SIGCOMM HotSDN Workshop*, August, 2014.

[20] "ONOS: Open Network Operating System", https://github.com/opennetworkinglab/onos.

[21] Heller B., Sherwood R., McKeown N., "The Controller Placement Problem", *Proceedings of the first workshop on Hot topics in software-defined networks*, ACM, 2012.

[22] Chinneck J.W., "Practical optimization: a gentle introduction", 2012, https://sce.carleton.ca/faculty/chinneck/po.html.

[23] Bolosky W., et al., "Paxos Replicated State Machines as the Basis of a High-Performance Data Store", *Proceedings of the NSDI*, 2011.

[24] Knight S., et al., "The internet topology zoo", http://www.topology-zoo.org.

[25] "Rocketfuel: An ISP Topology Mapping Engine", https://research.cs.washington.edu/networking/rocketfuel.

Pashkov V. N., "Fault-Tolerance Distributed Control Plane for Software-Defined Networks", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 101–121.

DOI: 10.18255/1818-1015-2019-1-101-121

Abstract. The architecture of the high availability distributed control plane for SDN/OpenFlow networks are considered. High availability is achieved by redundancy of controller instances, active switch-controller communications, computing resources and tools for a controller instance failure and overloading detection and recovery. The proactive backup controller allocation algorithm which allows to minimize the time to repair in the case of a single controller instance failure is discussed. The algorithm for controller load-balancing allows dynamically reconfigure the control plane with a minimum number of switch control transfer operations to avoid controller instance overloading. The initial experimental results of the proposed algorithms for the HA distributed SDN control plane are described.

Keywords: software-defined networking, SDN, distributed control plane, DCP, fault tolerance, load-balancing, OpenFlow, data plane, network architecture

On the authors:
Vasily N. Pashkov, Programmer, orcid.org/0000-0001-5783-4557
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: pashkov@lvk.cs.msu.su