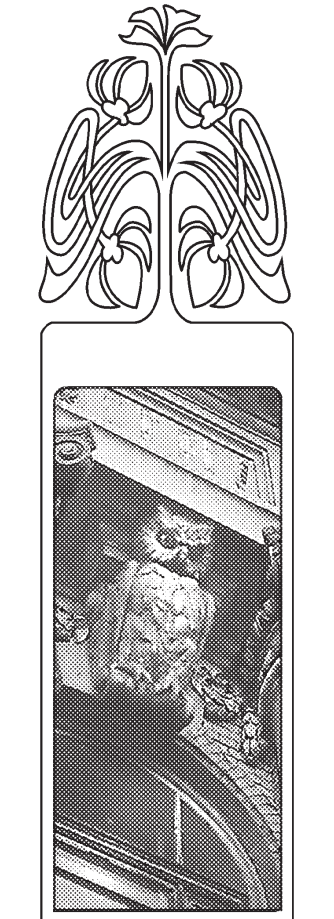

НАУЧНЫЙ ОТДЕЛ

ИНФОРМАТИКА

УДК 004.75

КОНВЕРГЕНТНЫЕ И ГИПЕРКОНВЕРГЕНТНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

В. М. Соловьев

Соловьев Владимир Михайлович, кандидат технических наук, доцент кафедры математической кибернетики и компьютерных наук, начальник Поволжского регионального центра новых информационных технологий, Саратовский национальный исследовательский государственный университет имени Н. Г. Чернышевского, 410012, Россия, Саратов, Астраханская, 83, svm@sgu.ru

В работе рассмотрены вопросы построения гиперконвергентных вычислительных систем и их функционирование на основе программно-конфигурируемой сети. Приведены особенности протокола OpenFlow и технологические решения, переносящие управление программно-конфигурируемой сетью на выделенный контроллер (сервер). Предложена графовая модель управления ресурсами гиперконвергентной вычислительной системы, отвечающая требованиям заданного качества обслуживания, с одной стороны, и экономическим требованиям, с другой. На основе предложенной модели рассмотрен вариант реализации жадного алгоритма управления конвергентной инфраструктурой по протоколу OpenFlow, осуществляющего назначения запросов на физические ресурсы, используя программное обеспечение контроллера. Показаны преимущества многопотоковой маршрутизации, реализованной с среде гиперконвергентной инфраструктуры, используя для ее описания минимального дерева Штейнера. Рассмотрены вопросы надежности и безопасности гиперконвергентных вычислительных систем, делающих большую часть современных угроз неактуальными. В работе приведены возможности импортозамещения и перспективы перехода на сетевую инфраструктуру, ориентированную на контент.

Ключевые слова: конвергентная инфраструктура, гиперконвергентные системы, импортозамещение, программно-конфигурируемая сеть (ПКС), software defined networks (SDN), протокол OpenFlow, виртуальный центр обработки данных (ВЦОД), планирование вычислительных ресурсов, жадные алгоритмы, Service Level Agreement (SLA), многопотоковая маршрутизация, Multi-Threaded Routing (MRT), Quality of Service (QoS), DDoS-атаки, перехват данных, Information Centric Networking (ICN).

DOI: 10.18500/1816-9791-2018-18-1-84-100

© Соловьев В. М., 2018

ВВЕДЕНИЕ

Конвергентные вычисления — это сближение различных методов обработки данных с целью сведение их к одной системе. В этой связи в информационные технологии (Information Technology, IT) был введен термин «конвергентная инфраструктура» (Convergent Infrastructure, CI), за которым кроется идея объединения средств хранения данных, вычислительных и сетевых ресурсов в единый пул облачных сервисов. Развитием такой инфраструктуры стали гиперконвергентные системы, переводящие конвергентные вычисления на следующий уровень. Гиперконвергентная инфраструктура (HyperConvergent Infrastructure, HCI) предлагает CI на основе масштабируемых программно-определяемых технологий. Таким образом, создаются архитектуры и платформы, которые гораздо более просты в управлении, масштабировании и обслуживании. При этом сети, вычислительные мощности и хранилища данных объединяются в одно целое с помощью программных средств, а управление ими происходит через общий контроллер (сервер), обслуживаемый одним системным администратором. Катализаторами развития этого нового IT мейнстрима стали финансовый кризис и идея импортозамещения [1].

Основой HCI является современная сетевая архитектура, комбинация стандартов, топологий и сетевых протоколов, необходимых для создания работоспособной распределенной вычислительной системы. Самая популярная и массовая в настоящее время сетевая архитектура Ethernet, по мнению членов глобальной исследовательской сети PlanetLab, устарела. В ближайшем будущем она не сможет поддерживать на должном уровне новые телематические сервисы и растущие запросы пользователей сети. Попытки изменить существующие архитектуры предпринимались довольно часто. Однако внесение принципиальных изменений в существующую сетевую архитектуру является весьма трудным делом, так как требует привлечения производителей телекоммуникационного оборудования. Кроме того, средства построения современных сетей являются проприетарными, закрытыми для изменений со стороны владельцев сетей и научной общественности. Практически всегда при изменении архитектуры переход с оборудования одного производителя на оборудование другого — сложная задача, требующая больших финансовых ресурсов.

Решить эту проблему предложила основанная в 2007 г. компания Nicira, которая одна из первых начала разработки программно-конфигурируемых сетей (ПКС, SDN, Software Defined Networks). Она предложила разделить уровни управления (Control Plane) и передачи данных (Data Plane), которые в современных сетях функционируют совместно, что делает контроль и управление сетью очень сложным. Идея разделения заключалась в «перехвате» управления таблицами коммутации (маршрутизации), что обеспечивало произвольное управление поведением и производительностью сетевого оборудования, а также параметрами передаваемых потоков данных в масштабах всех сетей Ethernet. Кроме того, независимость от оборудования производителя обеспечивал новый сетевой протокол OpenFlow. Такой подход основывался на специальном программном обеспечении, работающем на обычном компьютере, контролируемом администратором сети. Это позволило использовать вместо проприетарных средств построения сетей программное обеспечение с открытым кодом (open source) и заменить управление отдельным экземпляром сетевого оборудования управлением всей сетью. Таким образом, можно осуществлять виртуализацию

на уровне не отдельных узлов сети, а всей сети в целом, создав интеллектуальный программно-управляемый интерфейс между сетевыми приложениями и транспортной средой сети (рис. 1).

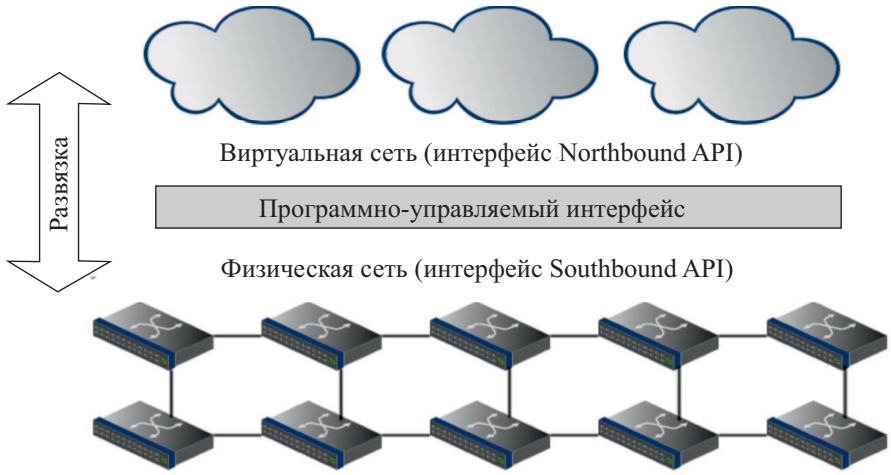

Рис. 1. Архитектура программно-коммутируемой сети
Fig. 1. The architecture of a software-switched network

Программно-управляемый интерфейс работает на границе сети и управляется распределенными кластерами контроллеров. Он позволяет динамически, без изменения адресов и нарушения рабочих процессов создавать виртуальные сети со всеми свойствами и сервисами реальных сетей. Независимость от оборудования такой архитектуры обеспечивает полную аппаратную межплатформенность, что делает возможным стабильное функционирование виртуальной сети даже в процессе замены оборудования.

Технологические решения SDN, предлагающие отделить Control Plane и Data Plane, переносят управление сетью на выделенный контроллер SDN. Такой подход включает три основные компоненты (рис. 1): контроллер SDN, обеспечивающий программно-управляемый интерфейс, API интерфейс взаимодействия с физической сетью (Southbound API) и API интерфейс взаимодействия с виртуальной сетью (Northbound API). Контроллер SDN является «главным вычислителем» ПКС и отвечает за построение топологии сети, программирование сетевых устройств и управление всей сетью. В качестве основного протокола контроллер SDN использует OpenFlow. Интерфейс Southbound API обеспечивает взаимодействие с реальными сетевыми устройствами. Интерфейс Northbound API обеспечивает интерпретацию бизнес-логики в сетевые инструкции, что позволяет администраторам гибко управлять сетевыми сервисами, исходя из требований приложений. Такая архитектура (см. рис. 1) позволяет отделить управление сетью от функций форвардинга данных (disaggregation). В связи с этим каждый из уровней может рассматриваться по отдельности, а не как единая интегрированная система. Это предоставляет приложениям более подробную и консистентную информацию о состоянии сети. Таким образом, приложения, предоставляющие определенный уровень абстракции сети, могут напрямую взаимодействовать с SDN контроллером, запрашивая сетевые сервисы посредством API. Контроллеры и коммутаторы сети работают каждый в своем сетевом пространстве и представляют собой отдельные виртуальные машины, подключенные

в сеть через интерфейсы Ethernet. При этом SDN контроллер интерпретирует инструкции и запросы от приложений в инструкции для сетевых устройств. Обратное же взаимодействие включает сбор информации о сети и отправку её SDN контроллерам, тем самым абстрагируясь от уровня сетевой инфраструктуры для приложений. Роль сетевых устройств сводится лишь к функции перенаправления данных согласно инструкциям SDN контроллера. Основное преимущество ПКС заключается в возможностях предоставления приложениям абстрактной сетевой инфраструктуры, что делает сеть более «интеллектуальной». Такая сеть имеет возможность самоанализа и принятия решений о форвардинге трафика, исходя из требований приложений и загрузки сети в режиме реального времени.

1. ПРОТОКОЛ OPENFLOW

Идеи, положенные в основу протокола OpenFlow, появились задолго до ПКС и на их основе пытались решить такие же задачи, что и сегодня. Эти идеи основаны на отделении управления сетевыми устройствами от самих устройств, используя модель из «вычислителя» сети и множества «простых аппаратных» сетевых устройств, отвечающих только за форвардинг. Для реализации этого используется механизм программирования правил коммутации. Таким механизмом и является протокол OpenFlow, рассматриваемый как набор инструкций (протокол программирования), и как архитектурный подход. При этом весь функционал, заложенный в OpenFlow, поддерживается на уровне оборудования. То есть снижаются требования к функционалу оборудования, оставляя ему лишь простые функции Data Plane, а функции Control Plane выносятся на отдельное устройство — SDN контроллер (рис. 2).

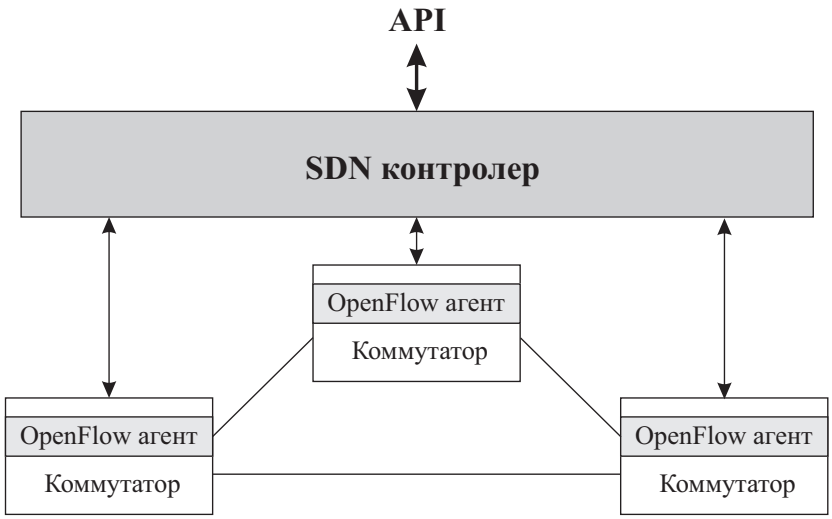

Рис. 2. Структура SDN контроллера
Fig. 2. Structure of the SDN controller

Контроллер использует протокол OpenFlow, чтобы подключиться к коммутаторам (маршрутизаторам) и запрограммировать их таблицы форвардинга, рассчитывая лучшие маршруты для трафика приложений. Назначение OpenFlow агента — принять команды от контроллера и сформировать таблицу, на основе которой коммутатор (маршрутизатор) решает, куда направить пришедший пакет. Структура Data Plane коммутатора приведена на рис. 3.

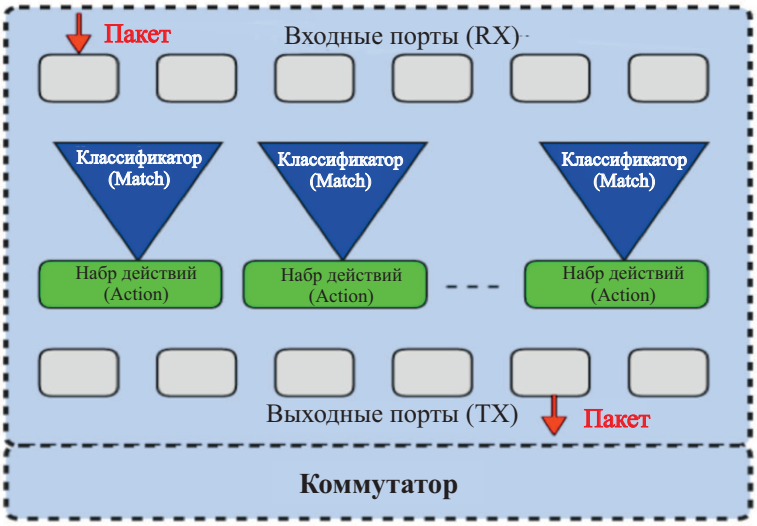

Рис. 3. Структура Data Plane коммутатора
Fig. 3. Structure of the Data Plane Switch

Порты OpenFlow выполняют те же функции, что и порты обычного коммутатора [2]. Пакеты, приходящие на OpenFlow-порт (Ports), классифицируются по потокам (Flow) в таблицах потоков (Flow tables) с помощью классификаторов (Match) [3]. Каждый поток состоит из набора действий (Actions), которые применяются к каждому пакету и соответствуют определенному правилу. При поступлении пакета на OpenFlow-порт из него извлекаются метаданные. Затем они сравниваются с записями в таблице потоков, где каждая запись в таблице имеет поле «protocol», по которому и выполняется сравнение.

Результату сравнения присваивается приоритет, по которому выбирается «победитель». Запись с наивысшим приоритетом (победитель) определяет то, как дальше будет обрабатываться пакет. Эту обработку определяет классификатор в соответствии с Match-правилом, соответствующим определенному действию. Эти действия (Actions) могут быть: отправить в порт N, отбросить пакет, отправить на контроллер и т. д. Если пакеты не соответствуют ни одному правилу, то их отправляют на контроллер. Классификатор может использовать любую комбинацию полей заголовка пакета (сравнение: по MAC-адресам, VLAN- и MPLS-меткам, IPv4- и IPv6-адресам, TCP- и UDP-портам и т. д.). После сравнения становится возможным выполнение ряда действий, например, использование протокола OpenFlow для реализации функций NAT.

В последних версиях протокола OpenFlow широко используется механизм групп (Group), который улучшает форвардинг. Группа включает набор подмножеств, а каждое подмножество — набор вышеуказанных действий. Улучшение форвардинга заключается в «группировке действий»: например, отправку подмножеств из группы для балансировки нагрузки сети и т. д. Другим направлением развития протокола OpenFlow является использование механизма TTP (Table Type Patterns) [3]. Механизм TTP базируется на двух моментах. Во-первых, OpenFlow агент на коммутаторе конфигурирует Data Plane по инструкциям, полученным от контроллера. Эти инструкции представляют собой пайплайн (pipeline) — определенную последовательность действий, совершаемых с пакетом на коммутаторе. Однако некоторые инструкции коммутатор не может выполнить из-за особенностей функционала чип-

сета. Поэтому предварительно при инициализации необходимо определить, какие пайплайн поддерживаются, а какие нет. Это согласование нужно провести между коммутатором и контроллером до того момента, как они начнут работать вместе. Во-вторых, при инициализации необходимо определить функционал OpenFlow, поддерживаемый контроллером и коммутатором. Затем, если контроллер и коммутатор «договорились», они используют только взаимно ограниченный функционал OpenFlow для реализации поддерживаемого пайплайна. Такой подход исключает неожиданное поведение коммутаторов в ответ на получение неподдерживаемых пайплайн. На практике нет необходимости поддерживать все возможные пайплайны OpenFlow, а только совместимые. Такое ограничение и получило название TTP, которое фактически содержит информацию о количестве таблиц, типах сообщений и о том, какие действия предпринимаются с пакетом при переходе из одной таблицы в другую. В настоящее время ПКС используют только контроллеры, поддерживающие TTP.

2. УПРАВЛЕНИЕ HCI ПО ПРОТОКОЛУ OPENFLOW

По сути, HCI — виртуальный центр обработки данных (ВЦОД) — виртуальная инфраструктура, которая позволяет создавать виртуальные машины, виртуальные хранилища данных, виртуальные коммутаторы и маршрутизаторы, а также виртуальные каналы связи. Основная задача ВЦОД — принимать запросы (тенанты) пользователей и актуализировать их с помощью средств виртуализации в сетевой топологии. В основе механизма управления ресурсами ВЦОД лежит расширенная для задач HCI модель [4]. Сетевая топология представлена графом $T = (C \cup M \cup K \cup L)$, где C — множество вычислительных узлов, M — множество хранилищ данных, K — множество коммутационных элементов сети, L — множество каналов связи. На каждом множестве определены векторы скалярного аргумента, задающего параметры: вычислительных узлов — $c \in C$, хранилищ данных (памяти) — $m \in M$, коммутационных элементов — $k \in K$ и каналов связи — $l \in L$ соответственно:

$$\begin{aligned} fct(c) &= (ct_1(c), ct_2(c), \ldots, ct_n(c)), \\ fmt(m) &= (mt_1(m), mt_2(m), \ldots, mt_n(m)), \\ fkt(k) &= (kt_1(k), kt_2(k), \ldots, fk_n(k)), \\ flt(l) &= (lt_1(l), lt_2(l), \ldots, lt_n(l)). \end{aligned} \tag{1}$$

Ресурсы ВЦОД заданы графом $R = (V \cup S \cup D)$, где V — множество приложений, развернутых на виртуальных машинах, S — множество виртуальных хранилищ данных, D — множество каналов связи между виртуальными машинами и хранилищами данных. На каждом множестве также определены векторы скалярного аргумента, задающего параметры: виртуальных машин — $v \in V$, виртуальных хранилищ данных — $s \in S$, каналов связи, включающих коммутационные элементы и обеспечивающих требуемое качество сервиса (SLA) — $d \in D$ соответственно:

$$\begin{aligned} fvr(v) &= (vr_1(v), vr_2(v), \ldots, vr_n(v)), \\ fsr(s) &= (sr_1(s), sr_2(s), \ldots, sr_n(s)), \\ fdr(d) &= (dr_1(d), dr_2(d), \ldots, dr_n(d)). \end{aligned} \tag{2}$$

Параметры (2), обеспечивающие SLA, совпадают с соответствующими параметрами (1) и представляются отображением запросов ресурсов на топологию HCI:

$$O : R \to T \cup \{\oslash\} = \{V \to C \cup \{\oslash\}, \quad S \to M \cup \{\oslash\}, \quad D \to K \cup \{\oslash\}, \quad L\{\oslash\}\}. \tag{3}$$

Запросы ресурсов из выражения (3) определяют три типа отношений между параметрами запросов r_i и физическими ресурсами t_i, исходя из топологии HCI: запрашиваемые ресурсы соответствуют ресурсам, определяемым топологией $r_i = t_i$; перегрузка физических ресурсов $r_i > t_i$, нарушающая SLA; недогруженность физических ресурсов $r_i < t_i$, требующая реконфигурации топологии по экономическим соображениям. В последнем случае доступные ресурсы могут быть представлены остаточным графом $T_res = (\cup M \cup K \cup L)$, который переопределяет параметры в следующем виде:

$$fct_{res}(c) = fct(c) - \sum_{v \in V} fvr(v), \quad fmt_{res}(m) = fmt(m) - \sum_{s \in S} fsr(s),$$
$$fkt_{res}(k) = fkt(k) - \sum_{d \in D} fvr(v), \quad flt_{res}(l) = flt(l) - \sum_{d \in D} fdr(d). \tag{4}$$

Выполнить требования SLA, с одной стороны, и экономические требования, с другой, позволяет автоматизированная миграция структур HCI под управлением контроллеров по протоколу OpenFlow. Миграция выполняется и когда невозможно по запросу назначить виртуальное хранилище, а данные заносятся на несколько физических хранилищ. В этом случае часть приложений может работать с виртуальным хранилищем данных, а часть с данными, располагающимися в другом физическом хранилище. В соответствии с планом миграции перемещение виртуальных структур должно отвечать следующим требованиям: при перемещении SLA не нарушается; все перемещения выполняются в заданные временные интервалы, что обеспечивает автоматизированная работа контроллеров. Исходными данными для миграции являются: множество поступающих запросов $Z = \{R_i\}$, множество выполненных запросов $W = \{R_i\}$, граф остаточных ресурсов T_{res}, ограничение на время миграции τ. При миграции в граф запрашиваемых ресурсов R добавляется новая вершина s' и виртуальный канал связи между вершинами s и s'.

Управление ВЦОД по протоколу OpenFlow осуществляется на основе жадного алгоритма назначения запросов на физические ресурсы, используя программное обеспечение контроллеров (серверов), в основе которого лежит выражение (4). В качестве критерия оптимизации применяется наиболее компактное размещение элементов запроса (2) с точки зрения ресурсов. Аналогичный подход широко используется в центрах обработки данных и провайдерами облачных сервисов [5,6]. Качество управления ВЦОД будет зависеть от выбора жадных критериев: очередного запроса — K_R, виртуального узла — K_V, физического узла — K_C. Критерии K_V и K_C основываются на функции стоимости, определяемой как взвешенная сумма требуемых параметров с учетом дефицита ресурса — $d(i) = (\sum_R \sum_{e \in R} r_{r,i} - \sum_{c \in C} r_{c,i}) / \sum_R \sum_{e \in R} r_{e,i}$. Тогда функция стоимости назначения элемента «е» будет иметь вид: $r(e) = \sum_{i=l}^{n} d(i) r_{e,i}$, где выбранный элемент HCI характеризуется вектором значений требуемых параметров ресурса $(r_{e,1}, r_{e,2}, \ldots, r_{e,n})$. Величина дефицита ресурса вычисляется как частное от деления, разницы между общим значением требуемого параметра ресурса по всем запросам и значениям имеющихся физических ресурсов по данному параметру, на общую сумму требуемых ресурсов. Тогда функция стоимости будет определяться как взвешенная сумма требуемых параметров ресурса с учетом их дефицита. В соответствии с критерием K_V выбирается виртуальный элемент HCI, у которого функция стоимости максимальна. Это делается для того, чтобы сначала назначить самые

дефицитные по ресурсам элементы, а затем все остальные виртуальные элементы. В соответствии с критерием K_C выбирается физический элемент HCI, у которого функция стоимости минимальна. Это делается для того, чтобы достичь максимальной утилизации (загрузки) вычислительных ресурсов. В соответствии с критерием K_R выбирается запрос, взвешенная сумма запрашиваемых ресурсов которого максимальна.

Общая схема алгоритма управления будет выглядеть следующим образом.

1. Планировщик анализирует пришедшие запросы ресурсов $Z = \{R_i\}$.

2. Если множество $\{R_i\} \notin \oslash$ непусто, то выбирается очередной запрос R_i в соответствии с жадным критерием K_R, в противном случае, работа алгоритма завершается.

3. Из элементов запроса R_i формируется множество виртуальных узлов $U = \{V \cup S\}$, в противном случае, при невозможности сформировать множество виртуальных узлов U выполняется переход на шаг 14.

4. Из сформированного множества виртуальных узлов U выбирается очередной элемент N в соответствии с жадным критерием K_V и помещается в очередь Q элементов, ожидающих назначение.

5. Из элементов C_i формируется множество физических узлов $\{C_i\} \notin \oslash$, на которые может быть назначен элемент N на основании корректного выполнения отображения (3). В противном случае, при $\{C_i\} \in \oslash$ вызывается процедура ограниченного перебора.

6. Из сформированного множества физических узлов в соответствии с жадным критерием K_C выбирается физический ресурс и переопределяются значения параметров физических ресурсов в соответствии с функциями (4).

7. Выбираются все виртуальные каналы D_i, связывающие элемент N с еще неназначенными элементами запроса R_i.

8. Сортируется множество каналов $\{D_i\} \notin \oslash$ по величине пропускной способности в возрастающем порядке.

9. Выбирается из множества каналов $\{D_i\}$ виртуальный канал L_i, обеспечивающий кратчайший маршрут, связывающий элемент N с элементом запроса R_i. В противном случае, при невозможности проложить маршрут вызывается процедура назначения виртуального канала на физическом ресурсе и переопределяются значения параметров физических ресурсов в соответствии с функциями (4).

10. Добавляются в очередь Q виртуальные узлы, связанные с N в соответствии с порядком связывающих их виртуальных каналов из отсортированного множества $\{D_i\}$.

11. Удаляется N из U и Q.

12. Если Q непусто, то выполняется переход к шагу 4.

13. Если U непусто, то выполняется переход к шагу 3, в противном случае, переход к шагу 1.

14. Отменяются все назначения элементов запроса R_i, удаляется запрос из множества $Z = \{R_i\}$ и выполняется переход к шагу 2.

В алгоритме используются две процедуры, ограниченного перебора и назначения виртуального канала на физическом ресурсе, описанные в [5]. Процедура ограниченного перебора вызывается при невозможности назначить очередной виртуальный узел N из множества запросов $\{R_i\}$ ни на один физический ресурс. Она анализирует подмножество множества физических узлов $\{C_i\}$ графа физических ресурсов.

Мощность подмножеств определяется заданной глубиной перебора, а количество просматриваемых подмножеств ограничено. Просматриваются только те подмножества, у которых суммарное количество остаточных ресурсов узлов достаточно для назначения текущего элемента N. Процедура обеспечивает выполнение шага 5 при изменении (подборе) глубины перебора и количества просматриваемых подмножеств. Процедура назначения виртуального канала на физическом ресурсе вызывается при невозможности проложить маршрут, связывающий виртуальным каналом элемент N с элементом запроса R_i. Поиск маршрута выполняется на основе модифицированного алгоритма Дейкстры [7] так, что в него могут входить только коммутационные элементы и каналы связи физической сети, для которых выполняются соотношения корректности отображения (3). При невозможности назначения виртуального канала, подключающего элемент хранения, осуществляется поиск хранилища, которое имеет ресурсы для создания реплики элемента хранения (реплика требует ресурсы, равные ресурсам элемента хранения). Все отобранные для создания реплики хранилища рассматриваются в порядке возрастания суммарной длины маршрутов. Кроме того, рассматривается возможность создания канала связи l, обеспечивающего пропускную способность и требуемую интенсивность потока данных, к реплике. В случае невозможности канала связи l обеспечить требуемые параметры рассматривается следующий вариант на размещение реплики. В конечном итоге создается маршрут, обеспечивающий связность между элементом N и репликой. Положительный результат обеспечивается изменением параметров маршрута и канала связи. Аналогичная техника в отношении виртуальных машин хорошо известна и рассмотрена, например, в работе [8].

Таким образом, алгоритм управления NCI по протоколу OpenFlow позволяет планировать вычислительные ресурсы, ресурсы хранения данных и коммуникацирнные ресурсы самоорганизующейся облачной платформы (ВЦОД) с соблюдением SLA и использованием технологических решений SDN. Он допускает рациональное использование физических ресурсов за счет устранения их сегментации, применяя миграцию виртуальных ресурсов. Алгоритм позволяет администрировать гиперконвергентную вычислительную систему, задавая политики маршрутизации потоков данных и используя сетевые функции управления виртуальными и физическими устройствами различных производителей, поддерживающих протокол OpenFlow. Предлагаемое решение позволяет объединять различные сети, управляемые по протоколу OpenFlow, и эффективно передавать потоки данных между ними.

3. МНОГОПОТОКОВАЯ МАРШРУТИЗАЦИЯ В HCI

Требования к качеству сервиса в вычислительных системах постоянно растут, так как современные сетевые приложения требуют высокой производительности — моделирование в реальном времени, телемедицина (дистанционная хирургия); высокой производительности и малого джиттера — видеоконференции, IP-телефония, IP-телевидение; высокой производительности и низкой частоты ошибок — банковские системы, платежные терминалы и т. д. Этим требованиям отвечает многопотоковая маршрутизация, реализованная в HCI, которая обеспечивает высокое качество сетевых сервисов (QoS) и прежде всего высокую производительность путем распараллеливания информационных потоков. Эта возможность предоставляется SDN, так как позволяет управлять QoS с помощью контроллера, обеспечивающего глобальное видение топологии сети и возможности построения нескольких непересека-

ющихся маршрутов на основе многопотокового транспортного протокола Multipath TCP (MPTCP) в рамках модифицированного протокола управления передачей TCP (Transmission Control Protocol). В этом случае SDN контроллер анализирует TCP заголовок, детектирует открытие нового MPTCP соединения и открытие нового подпотока для известного MPTCP соединения, прокладывает кратчайшие маршруты. Причем маршруты одного соединения не пересекаются. Используемая ранее традиционная маршрутизация на основе IP адресов для борьбы с перегрузками требовала бы наращивания ресурсов: широкополосных каналов связи и быстрых коммутаторов, а это без управления QoS снижало бы утилизацию ресурсов. Чтобы обеспечить все запросы ресурсами сеть без управления качеством сервиса должна была бы ориентироваться на наибольшие запросы. В HCI коммутаторы оперируют только портами и адресами заголовков канального, сетевого и транспортного уровней. Контроллер же реализует различные маршруты следования трафика, наблюдая за сетью, он выполняет роль арбитра, самостоятельно определяющего количество подпотоков. Он динамически их ограничивает и вырабатывает оптимальное распределение ресурсов, т. е. решает задачу многокритериальной оптимизации. В настоящее время образцов таких контроллеров выпускается более 30, имеется и российский контроллер Runos, а для образовательных целей рекомендуется использовать контроллер Pox.

Многопотоковая маршрутизация использует сервисы MPTCP, которые для одного соединения дают возможность одновременно передавать данные несколькими маршрутами. Такая маршрутизация в сети может быть представлена графом $G = (V, E)$, где V — множество вершин (коммутационных узлов) в соединении, через которые проходят соответствующие подпотоки, а E — множество ребер, которые входят в n маршрутов. В этом случае сеть передачи данных может быть описана минимальным деревом Штейнера: $G_C(n) = (V_C(n), E_C(n))$, которое включает в себя все вершины $V_C(n) \in V$ и ребра $E_C(n) \in E$. Тогда задача маршрутизации (алгоритм маршрутизации) сводится к задаче Штейнера на графе, т. е. к отысканию во взвешенном графе дерева, покрывающего выделенное подмножество вершин и ребер. При этом вес определяется исходя из длины ребра, а суммарная длина ребер $\sum_n |E_C(n)|$ должна быть минимальной. Такое дерево будет представлять собой скопление сетей $c_i = (v_i, e_i)$, где i — количество сетей, соответствующее подпотокам, а $e_i \in E$, $v_i \in V$. Причем скопление сетей будет определять производительность и отсутствие перегрузок при маршрутизации, что в конечном итоге будет управлять QoS. Многопотоковая маршрутизация подразумевает три аспекта: реализации архитектуры сети на основе высокоуровневых решений Multipath TCP; реализации механизмов подавления перегрузки на основе алгоритма маршрутизации при отсутствии взаимодействия разных маршрутов (без их пересечения); взаимодействие MPTCP с приложениями через программный интерфейс (API). Работа по модифицированному протоколу управления передачей может происходить по следующему сценарию.

1. Для приложений, где отсутствует поддержка MPTCP, протокол ведет себя как обычный протокол TCP, и приложение начинает работу как обычно с формирования TCP-сокета.

2. Соединение MPTCP устанавливается как и обычное TCP-соединение.

3. При доступности дополнительных маршрутов (наличия сетевых ресурсов) создаются дополнительные TCP-сессии, называемые подпотоками. Они комбинируются с существующими сессиями, которые представляют собой отдельные соединения приложений.

4. Протокол MPTCP идентифицирует несколько маршрутов за счет наличия у сетевого узла нескольких адресов. Комбинации этих адресов позволяет сформировать дополнительные маршруты.

5. Формирование и конфигурирование подпотоков осуществляется с помощью управления маршрутизацией.

6. Модифицированный протокол управления передачей добавляет порядковые номера уровня соединения, чтобы позволить сборку сегментов, доставляемых по разным маршрутам с разными задержками.

7. Подпотоки завершаются как TCP-соединения посредством четырехтактного диалога FIN.

Основные функции многопотоковой маршрутизации ложатся на SDN-контроллер и заключаются в определении им маршрутов передачи пакетов, а также в управлении сетевыми устройствами, обрабатывающими и балансирующими потоки данных. С точки зрения QoS маршруты должны отвечать следующим требованиям: должна выполняться равномерная загрузка (балансировка) сети; маршруты должны прокладываться с учетом пропускной способности ресурсов и требований сетевых приложений; маршруты должны иметь наименьший вес в терминах MPTCP. Реализация этого обеспечивается алгоритмом, реализуемым на основе минимального дерева Штейнера.

4. НАДЕЖНОСТЬ И БЕЗОПАСНОСТЬ HCI

Большое внимание в гиперконвергентных системах, активно развивающихся на основе SDN, уделяется не только качеству сетевых сервисов, но также обеспечению безопасности коммуникаций и надежности сети передачи данных. В HCI это становится возможным за счет гибких механизмов распределенного управления и использования нескольких экземпляров контроллеров [9]. Высокая надежность сети обусловлена самовосстановлением и автоматическим перераспределением потоков данных в соответствии с выбранными правилами. Кроме того, на надежности работы сети сказывается эффективная визуализация трафика (круглосуточный мониторинг) и возможность перестройки физической сети без прерывания ее работы на обслуживание, обнаружение и устранение отказов. Безопасность HCI повышается за счет полной изоляции виртуальных сред другу от друга.

Выше приводилось описание контроллера, где он представлялся центральным элементом HCI, позволяющим коммутационным узлам и приложениям обмениваться данными. Поэтому к нему должны быть предъявлены повышенные требования по безопасности и надежности. Ошибки контроллера проявляются как ошибки доставки сообщений: недоставка сообщения от управляемого им коммутатора отдельному приложению или сообщения от приложения коммутатору. В HCI пользователь ожидает, что при отказе или перегрузке одного элемента его функции будут перехвачены другим, причем смена элементов произойдет в пределах ожидаемого интервала времени. Если такой перехват не происходит либо занимает больше времени, это также является ошибкой контроллера. Следующие ошибки — это ошибки сетевых приложений, проявляющиеся в нарушении сформулированных приложением утверждений. Эти ошибки не зависят от приложений, могут быть легко обнаружены и устранены контроллером. Третий подкласс ошибок — это ошибки конкуренции, возникающие как несогласованная работа приложений. Чаще всего они проявляются в виде конкуренции: за производительность коммутаторов (при невозможности обработать боль-

шое количество сообщений, часть из них коммутатор сбрасывает); за последовательность выполнения управляющих сообщений приложений, так как коммутатор может изменять порядок обработки сообщений; за таблицы коммутаторов. Методы борьбы с ошибками вытекают из классических принципов контроля и диагностирования — обнаружение ошибки в ходе контроля (мониторинга) сети, локализации ошибки в ходе диагностирования и ее устранение (рис. 4).

Рис. 4. Вывод информации средствами анализа пакетов в Linux
Fig. 4. Information output by means of package analysis in Linux

Состояние сети в HCI можно представить как функцию $A = f(E, T_E) \to (C, T_C)$, $C = \{C^i\}$, $C^i = \{C^{i,j}\}$, где E — последовательность событий в сети (изменение состояния каналов связи), T_E — последовательность временных интервалов событий, C — последовательность состояний сети, T_C — последовательность временных интервалов состояний, C^i — состояние сети в i-й момент времени (временной срез состояния коммутаторов), $C^{i,j}$ — состояние таблицы j-го коммутатора в i-й момент времени. Поскольку с HCI одновременно могут работать N сетевых приложений, то можно говорить о множестве последовательностей состояний $C_k = f_k(E, T)$, $k = 1, \ldots, N$. Корректная работа приложений в сети регламентируется правилами, которые проверяют свойства сетевых пакетов, и если свойства удовлетворяют требованиям правил, то над пакетом коммутатор выполняет соответствующие действия. Поэтому состояние сети в HCI будет зависеть от применяемых правил $A = f(E, T_E) \to (C, T_C, R, T_R)$, где R — последовательность применяемых правил, T_R — последовательность временных меток правил.

Вопросы безопасности HCI актуальны еще и по причине своей особенности, привлекающей злоумышленников. Во-первых, это возможность управлять сетью с помощью программного обеспечения, которое часто имеет уязвимости. Во-вторых, централизованное управление сетью с помощью контроллера (сервера), когда, получив доступ к нему, можно скомпрометировать всю сеть через управляющее программное обеспечение. К основным угрозам, специфичным для HCI, обычно относят: использование вредоносного кода для перехвата данных, DDoS-атаки на контроллер, атаки на сетевые устройства внутри сети, подключение вредоносных устройств внутри сети и уязвимости программного обеспечения. Основными методами противодействия этим угрозам является исключение возможности компрометации контроллера, что не позволяет злоумышленнику управлять сетью. С этой целью обеспечиваются целостность контроллера и строгий механизм аутентификации доступа к нему, чтобы исключить внедрение в него нежелательной информации. Обычно для этого используются криптографические протоколы SSL/TLS, которые еще и обеспечивают защи-

щенную связь между контроллером и OpenFlow-устройствами. В HCI это позволяет исключить, например, компрометацию OpenFlow-коммутатора, находящегося в открытом доступе. Коммутатор теперь может быть очень простым устройством, весь «интеллект» которого располагается в хорошо защищенном контроллере.

Идеология HCI делает большую часть угроз неактуальными (например, отказ от традиционных протоколов маршрутизации), а часть механизмов защиты может быть реализована в контроллерах или узлах сети, которые легко сегментируются и изолируются. В сети средствами HCI можно создавать эффективные фильтры и при необходимости их динамически перестраивать, что обеспечивает быстрое реагирование на ситуации и предотвращать атаки. В настоящее время HCI может использовать фильтры, запрещающие или разрешающие пропуск датаграмм с широковещательным адресом назначения между внутренними и внешними сетями; пропуск датаграмм, направленных из внутренней сети во внешнюю сеть, но имеющих внешний адрес отправителя; пропуск датаграмм, прибывающих из внешней сети, но имеющих внутренний адрес отправителя; пропуск датаграмм с опцией «source routing»; пропуск датаграмм с ICMP-сообщениями между внутренней и внешней сетями; пропуск датаграмм с UDP-сообщениями, направленными в порты echo и chargen; использование TCP intercept для защиты от атак SYN; фильтрацию TCP-сегментов, выполняемую в соответствии с политикой безопасности (разрешать все сервисы, кроме запрещенных, или запрещать все сервисы, кроме разрешенных); обработку ICMP echo-запросов, направленных на широковещательный адрес; обработку ICMP-сообщений Redirect, Address Mask Reply, Router Advertisement, Source Quench; попытки открытого сканирования.

Перехват данных в простейшем варианте начинается с прослушивания сети, исключить которую можно в HCI с помощью сегментации сети коммутаторами. Сегментация позволит злоумышленнику перехватить только кадры того сегмента сети, к которому он подключен. Более сложный случай перехвата трафика в IP-сети использует протокол ARP. При этом рассылаются сфальсифицированные ARP-сообщения, которые заставляют MAC-адрес злоумышленника считать адресом своего собеседника. Здесь сегментация не является препятствием для ARP-атаки. В HCI для борьбы с такими атаками используют статические ARP-таблицы и базу данных соответствия MAC- и IP-адресов всех узлов сети. Причем правила безопасности позволяют автоматизировать процесс создания статических таблиц. Еще одним вариантом перехвата трафика может быть ложная маршрутизация, когда злоумышленник навязывает узлу сети свой адрес в качестве адреса маршрутизатора. Это обеспечивает перенаправление трафика на узел злоумышленника и последующую его компрометацию. Чаще всего ложная маршрутизация выполняется с помощью фальсифицированных ICMP-сообщений Redirect; конфигурирования атакуемого узла ICMP-сообщениями Router Advertisement или использования протокола DHCP; атака на протоколы маршрутизации с целью переключения требуемых маршрутов на злоумышленника, при использовании атаки на протокол BGP. Для противодействия перечисленным угрозам в HCI используют правила, исключающие имперсонацию. Это прежде всего оперативное применение правил, исключающих обработку соответствующих сообщений (например, сообщений Redirect); автоконфигурирование; при использовании аутентификации в TCP-сегментах, например, с помощью алгоритма MD5 [10].

Кроме перечисленных угроз перехват данных позволяет злоумышленнику опреде-

лить установленные в HCI правила и попытаться их обойти. При этом используется анализ перехваченных пакетов, позволяющий определить статус сети. Какие пакеты перехватывать и анализировать, определяется из того факта, что пакеты, удовлетворяющие требованиям правила, обрабатываются быстрее, чем если бы они анализировались контроллером на предмет, что с ними дальше делать. Отсюда следует, что анализу нужно подвергать пакеты, которые по сети проходят быстрее остальных.

Следующей наиболее распространенной угрозой для HCI является отказ в обслуживании (DDoS). Эта угроза связана с перехватом данных и реакцией контроллера на пакеты, неудовлетворяющие требованиям правил. Злоумышленник после анализа перехваченных пакетов выбирает те (по времени), на которые контроллер создает новые правила. Цель атаки заключается в том, чтобы заставить контроллер генерировать множество новых потоковых правил за короткий промежуток времени. В результате такой атаки коммутатору нужно обработать множество вариантов таблицы потоков данных. Критерий переполнения будет иметь следующий вид:

$$R_{flow\,rule} > n_{flow\,table}/t_{flow\,table},$$

где $R_{flow\,rule}$ — максимальное количество потоковых правил, генерируемых в секунду; $n_{flow\,table}$ — максимальное количество записей в таблице потоков коммутатора; $t_{flow\,table}$ — время записи нового потокового правила. В результате атаки создаётся множество правил обработки потока данных в коммутаторе, по которым они обрабатываются с такой интенсивностью, что коммутатор не успевает их обработать и отказывается обрабатывать новые правила из-за переполнения. Решением проблемы DDoS может быть хорошо продуманная стратегия фильтрации с использованием межсетевых экранов и динамическое управление пропускной способностью каналов связи. Управляемая пропускная способность позволит контроллеру и коммутаторам обмениваться сообщениями даже во время атаки и даст возможность заблокировать вредоносный трафик. В HCI можно управлять количеством новых потоковых правил за счет объединения потоков, которые требуются для управления трафиком. Однако следует помнить, что объединение разных потоков для обработки по одним и тем же правилам может привести к непредсказуемым последствиям. Контроллеры, используемые в HCI, имеют возможность автоматически обнаружить DDoS на основе анализа трафика и выявлять в нем характерные аномалии, так как он контролирует всю информацию о текущем состоянии сети.

Таким образом, обзор возможных уязвимостей сетей и методов противодействия им показал, что использование идеологии HCI при построении вычислительных систем позволяет сократить количество сетевых уязвимостей и эффективно реализовывать целый комплекс методов защиты от атак.

ЗАКЛЮЧЕНИЕ

Конвергентные и гиперконвергентные вычислительные системы можно рассматривать применительно к любым вычислительным платформам (аппаратным, программным, облачным, нейроморфным, квантовым и т. д.), которые могут обеспечивать пользователю доступ к различным сервисам. Они должны обладать удобным интерфейсом и поддерживать несколько уровней инфраструктуры, обязательными из которых должны быть уровни безопасности, надежности, коммуникационных сервисов, обеспечивающих QoS для разных данных. Кроме того, сеть, лежащая в основе HCI, должна иметь возможности работы с разными видами терминалов (мобиль-

ными, десктопными, активными сетевыми, перспективными UX/UI и т. д.) и иметь единую платформу управления (контроллер, сервер) всем перечнем сервисов, приложений, аппаратных средств и каналов передачи данных. Причем канал передачи данных она должна выбирать в реальном масштабе времени, исходя из QoS и учитывая потребности приложений в производительности и характере трафика. Конвергентные технологии — это не конечная точка в развитии вычислительных систем нового поколения. Она уже сейчас позволяет реализовать на проверенной инфраструктуре HCI контент ориентированный подход, создать вычислительные системы, отходящие от парадигмы соединения конечных точек (end – to – end) к парадигме адресации по контенту или данным (Information Centric Networking, ICN). Эта парадигма предполагает организацию данных независимо от географического положения (сервер, хост) за счет распределенного сетевого кеширования. Ожидаемые преимущества от такого подхода — это более эффективное использования дорогих сетевых ресурсов, масштабируемость вычислительных систем, адаптивность их к меняющемуся QoS. Основой парадигмы являются примитивы publish/subscribe — опубликовать контент (сделать его доступным) и объявить об этом. Реализованы они в сетевой архитектуре, ориентированной на данные (Data Oriented Network Architecture, DONA). Работает это следующим образом: элемент такой сети получает запрос от себя подобном или от хоста. При этом может произойти два события. Если в кэше у элемента есть нужные данные, то он реализует запрос. Если контента нет, элемент DONA запрашивает себе подобных, у которых эти данные имеются. Получив ответ, он кэширует контент и реализует запрос. Этот механизм универсален и применим к любому протоколу, что образует глобальный единый механизм кеширования и доставки любого контента. Причем этот механизм поддерживается всеми узлами сети и ориентирован на всех пользователей, а не только на пользователей ICN. В такой сети обеспечивается безопасность контента, а не безопасность его доставки. Она основана на контентно ориентированной модели и построена на понятии «реноме», так как контент обязательно подписывается поставщиком, и потребители всегда его могут определить. Сетевая архитектура DONA хорошо взаимодействует с технологией blockchain, обеспечивая высокую надежность хранения и защиту контента. Вход в такую сеть защищен криптографически, и несанкционированный вход потребует огромных вычислительных ресурсов, пропорциональных размеру сети. Это же позволяет исключить ошибки человека или компьютера, пропущенные операции, несогласованный вход и т. д. В дальнейшем, эволюционируя, HCI будет использовать и другие перспективные сетевые технологии.

Библиографический список

1. *Орлов С.* Импортозамещение в ИКТ: взгляд производителей // Журн. сетевых решений / LAN. 2015. № 10. С. 48–50.
2. OpenFlow Switch Specification. Version 1.3.4. Open Networking Foundation. 2014. 171 p. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf (дата обращения: 04.07.2017).
3. OpenFlow Table Type Patterns. Version 1.0. Open Networking Foundation. 2014. 55 p. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/OpenFlow%20Table%20Type%20Patterns%20v1.0.pdf (дата обращения: 04.07.2017).
4. *Clos C.* A study of non-blocking switching networks // The Bell System Technical Journal. 1953. Vol. 32, № 2. P. 406–424. URL: http://ieeexplore.ieee.org/stamp/

stamp.jsp?arnumber=6770468 (дата обращения: 04.07.2017). DOI: 10.1002/j.1538-7305.1953.tb01433.x.
5. *Зотов И. А., Костенко В. А.* Алгоритм распределения ресурсов в центрах обработки данных с единым планировщиком для различных типов ресурсов // Изв. РАН. Сер. Теория и системы управления. 2015. № 1. С. 61–71. DOI: 10.7868/S000233881501014X.
6. *Meng X., Pappas V., Zhang L.* Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement // INFOCOM, 2010 Proceedings IEEE. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5461930 (дата обращения: 04.07.2017). DOI: 10.1109/INFCOM.2010.5461930.
7. *Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.* Introduction to Algorithms. Cambridge MA : MIT Press and McGrawHill, 2001. P. 595–601.
8. *Zhao M., Figueiredo R. J.* Experimental Study of Virtual Machine Migration in Support of Reservation of Cluster Resources // VTDC '07 Proceedings of the 2nd international workshop on Virtualization technology in distributed computing. 2007. P. 1–8. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5483380 (дата обращения: 04.07.2017). DOI: 10.1145/1408654.1408659.
9. *Смелянский Р. Л.* Концепция программно-конфигурированных сетей : от идеи до стандартизации // CONNECT! Мир связи : Наука. Бизнес. Управление. 2016. № 4. С. 62–67.
10. *Смелянский Р.* Настоящее и будущее SDN&NFV // Первая миля. 2016. № 3. С. 78–85.

Образец для цитирования:
Соловьев В. М. Конвергентные и гиперконвергентные вычислительные системы // Изв. Сарат. ун-та. Нов. сер. Сер. Математика. Механика. Информатика. 2018. Т. 18, вып. 1. С. 84–100. DOI: 10.18500/1816-9791-2018-18-1-84-100.

Convergent and Hyperconvergent Computing Systems

V . M. Solovyev

Vladimir M. Solovyev, https://orcid.org/0000-0003-3778-8201, Saratov State University, 83, Astrakhanskaya Str., Saratov, Russia, 410012, svm@sgu.ru

In the work the questions of construction of hyperconvergent computer systems and their functioning on the basis of a program-configurable network are considered. The features of the OpenFlow protocol and technological solutions that transfer control of the software-configurable network to a dedicated controller (server) are presented. A graph model of resource management of a hyperconvergent computer system is proposed that meets the requirements of a given quality of service on the one hand and economic requirements on the other. Based on the proposed model, an embodiment of a greedy algorithm for managing a converged infrastructure using the OpenFlow protocol and realizing requests for physical resources using the controller software is considered. The advantages of multithreading routing realized with the environment of hyperconvergent infrastructure are shown, using for its description the minimal Steiner tree. The issues of reliability and safety of hyperconvergent computing systems that make most of today's threats not relevant are considered. The paper shows the possibilities of import substitution and the prospects for switching to a network infrastructure, focused on content.

Key words: converged infrastructure, hyperconvergent systems, import substitution, software defined networks (SDN), OpenFlow protocol, virtual data processing center (VDPC), scheduling of computing resources, greedy algorithms, Service Level Agreement (SLA), Multi-Threaded Routing (MRT), Quality of Service (QoS), DDoS attacks, data interception, Information Centric Networking (ICN).

References

1. Orlov S. Import substitution in ICT: the view of producers. *Network Solutions Journal / LAN*, 2015, no. 10, pp. 48–50 (in Russian).
2. *OpenFlow Switch Specification. Version 1.3.4. Open Networking Foundation*. 2014. 171 p. Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf (Accessed 4 July 2017).
3. *OpenFlow Table Type Patterns. Version 1.0. Open Networking Foundation*. 2014. 55 p. Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/OpenFlow%20Table%20Type%20Patterns%20v1.0.pdf (Accessed 4 July 2017).
4. Clos C. A study of non-blocking switching networks. *The Bell System Technical Journal*. 1953, vol. 32, no. 2, pp. 406–424. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6770468 (Accessed 4 July 2017). DOI: 10.1002/j.1538-7305.1953.tb01433.x.
5. Zotov I. A., Kostenko V. A. Resource allocation algorithm in data centers with a unified scheduler for different types of resources. *Journal of Computer and Systems Sciences International*. 2015, vol. 54, no. 1, pp. 59–68. DOI: 10.1134/S1064230715010141.
6. Meng X., Pappas V., Zhang L. Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. *INFOCOM, 2010 Proceedings IEEE*. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5461930 (Accessed 4 July 2017). DOI: 10.1109/INFCOM.2010.5461930.
7. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. *Introduction to Algorithms*. Cambridge MA, MIT Press and McGrawHill, 2001, pp. 595–601.
8. Zhao M., Figueiredo R. J. Experimental Study of Virtual Machine Migration in Support of Reservation of Cluster Resources. *VTDC '07 Proceedings of the 2nd international workshop on Virtualization technology in distributed computing*, 2007, pp. 1–8. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5483380 (Accessed 4 July 2017). DOI: 10.1145/1408654.1408659.
9. Smelyansky R. L. The concept of software-configured networks: from idea to standardization. *CONNECT! Mir sviazi: Nauka. Biznes. Upravlenie* [CONNECT! The World of Connection: Science. Business. Control], 2016, no. 4, pp. 62–67 (in Russian).
10. Smelyansky R. Present and future of SDN&NFV. *Last Mile*, 2016, no. 3, pp. 78–85 (in Russian).

Cite this article as:
Solovyev V. M. Convergent and Hyperconvergent Computing Systems. *Izv. Saratov Univ. (N. S.), Ser. Math. Mech. Inform.*, 2018, vol. 18, iss. 1, pp. 84–100 (in Russian). DOI: 10.18500/1816-9791-2018-18-1-84-100.