МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М. В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 18

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

МАКС ПРЕСС

Москва – 2018

УДК 519.6+517.958
ББК 22.19
П75

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

Программные системы и инструменты : Тематический сборник /
П75 Под ред. Р. Л. Смелянского. – Москва : Издательский отдел факультета ВМК МГУ имени М. В. Ломоносова (лицензия ИД № 05899 от 24.09. 2001 г.); МАКС Пресс, 2018. – № 18. – 132 с.

ISBN 978-5-89407-594-5 (ВМК МГУ имени М. В. Ломоносова)
ISBN 978-5-317-06026-8 (МАКС Пресс)

Данный выпуск сборника составлен по материалам работ студентов, получивших рекомендации научных семинаров кафедры Автоматизации систем вычислительных комплексов, которую Л. Н. Королев создал и которой бессменно руководил, а также государственных аттестационных комиссий выпускников бакалавриата и магистратуры. Редколлегия сборника продолжает традицию издания сборника в память об этом выдающемся человеке. В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные проблемам современных компьютерных сетей, методам и средствам организации и управления облачными вычислениями, инструментальным средствам, обеспечивающим работу СРВ.

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

Ключевые слова: информационно-телекоммуникационные технологии, программно-конфигурируемые сети, OpenFlow-коммутатор, централизованный контроллер, беспроводные Wi-Fi сети, облачные вычисления, центр обработки данных, виртуальные ресурсы, виртуализация сетевых функций, облачная платформа, система реального времени, интегрированная модульная авионика, построение расписаний, свёрточные нейронные сети, эволюционные алгоритмы, задача оптимизации надёжности.

УДК 519.6+517.958
ББК 22.19

ISBN 978-5-89407-594-5
ISBN 978-5-317-06026-8

Оглавление

СБОРНИК
«Программные системы и инструменты»

От редколлегии:

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

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

Редколлегия

Pinaeva N.M., Antonenko V.A.

Inter-domain Communication Approach in MANO Platform[1]

Introduction

Network Function Virtualization (NFV) adds new capability to deploy network functions (NF) in cloud infrastructure thus NF becomes virtual network function (VNF). This technology decouples NF software from hardware it uses. The advantages it brings are: dynamic scaling, automatic NF configuration, efficient usage of physical resources etc.

Standards for NFV management and orchestration (NFV MANO) platforms' architecture are developed by ETSI NFV ISG [1]. The referenced architecture describes aspects of virtual infrastructure management and resources orchestration for network services (NS) that are represented as NF graph or chain provisioning.

NFV management and orchestration defines three main hierarchical virtualization layers [2]:

- Network Functions Virtualization Infrastructure (NFVI, virtual infrastructure) — hardware and software environment for VNF deployment. Virtual infrastructure usually includes virtual compute, storage and network resources.
- Virtualized Network Function (VNF) — implementation of NF which may be deployed and executed in virtual infrastructure.
- Network Service (NS) — composition of network functions, presented as forwarding graph, where vertices are VNFs and edges are links among them which point network traffic direction.

Network services are usually represented as Service Function Chains (SFC) [3]. SFC is an ordered set of NFs and restrictions on traffic that should flow through these NFs. Traffic restrictions are defined with traffic classifier which defines packets that should be handled with SFC. NFs from SFC are also called SFC nodes.

All of described entities have management modules, Virtualized Infrastructure Manager (VIM) for virtual infrastructure, Virtualized Network Function Manager (VNFM) for VNF and Network Functions Virtualization Orchestrator (NFVO) for NS. In case of multiple virtual infrastructure located in different points of presence (PoP) every

[1]This research is supported by the RFFI Foundation Grant N 18-07-01245, 2018

virtual infrastructure instance has its own VIM, we term every virtual infrastructure instance as virtual infrastructure domain or just domain. Network functions are no longer tied to hardware, therefore we can choose where to place it according to some requirements. Non-exhaustive list of potential benefits is:

- New VNF placement policies available (users' geographical requirements, other network services' VNFs placement, VNF's hardware requirements e.g. choose domain with virtual GPUs),
- QoS increase (e.g. place NS as close as possible to user to minimize latency),
- Increasing VNF's resilience (duplicating VNF instance in another virtual infrastructure domain, or migration),
- Optimizing virtual infrastructure resource utilization.

In distributed model of management and orchestration platform which has multiple VIMs, NFV orchestrator is able to choose virtual infrastructure domain to deploy VNF. This model provides discussed benefits and has architecture shown in Fig. 1 also presented by ETSI. In distributed architecture a new block WAN Infrastructure Manager (WIM) appears. It manages a new type of resource — WAN, and it is responsible for network connection between virtual infrastructure domains.

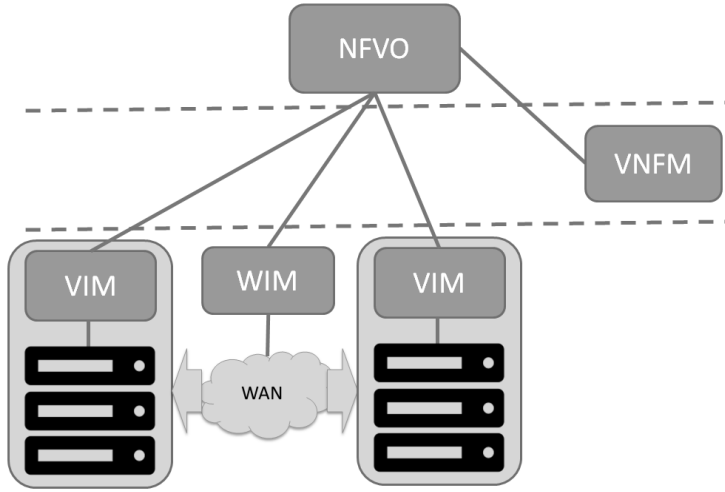

Fig 1. Distributed management and orchestration platform architecture

In this paper we present specification for distributed architecture and its implementation in our management and orchestration platform C2. In Section 1 we analyze platforms architectures described in related works, Section 2 describes designed architecture and its implementation. Section 3 contains experimental studies for developed system and Section 4 concludes this work.

1. Related work

ETSI management and orchestration architecture for distributed platform has a high level of abstraction and does not specify WIM interfaces well. Therefore we need to describe requirements for WIM and its interfaces.

Network connection between two virtual machines in different domains requires network resources in three network domains: internal virtual infrastructure domain network, virtual infrastructure domain gateway and WAN connecting two virtual infrastructure domains. SFC creation that has nodes in different domains consists of 3 steps (Fig. 2):

- VIM1 creates part of SFC which includes VNFs in its domain,
- VIM2 creates part of SFC which includes VNFs in its domain,
- WIM creates WAN part of SFC connecting VNFs from different domains.

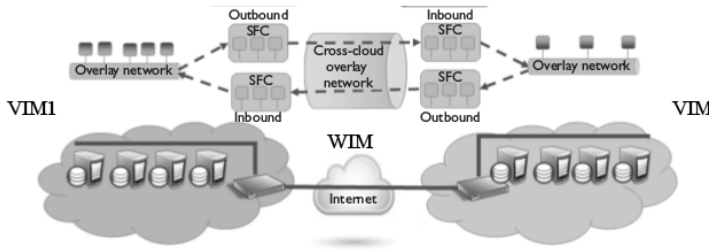

Fig 2. Distributed SFC scheme

SONATA [4] is an open-source system for management and orchestration. It consists of the following components (Fig.3):

- SONATA Service Platform (SP) — orchestration component responsible for NS lifecycle.

- WIM — component which manages WAN resources and creates connections between nodes located in different domains. All domain gateways are supposed to be based on Software-defined Networking (SDN).
- VIM — component that creates VNF instances and part of SFC within the virtual infrastructure domain.

WIM share virtual infrastructure gateway management with corresponding VIMs to instantiate SFC that has nodes in different domains.

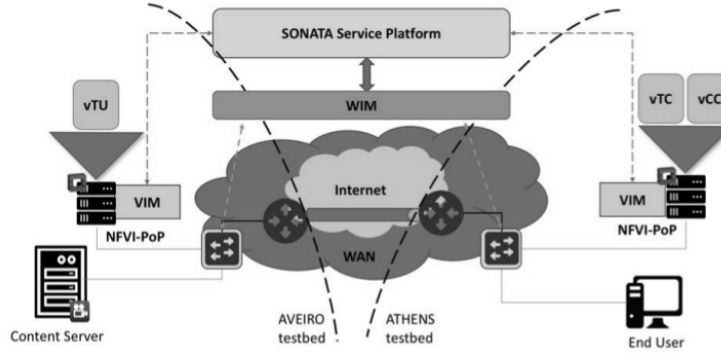

Fig. 3 Sonata architecture

Another project focused on NFV management and orchestration stack is T-NOVA [5]. It has architecture with following modules sharing distributed SFC creation:

- Vitrual infrastructure management includes VIM and WAN Infrastructure Connection Management (WICM, similar to WIM in ETSI). T-NOVA uses OpenStack [6] interfaces for virtual resources management and OpenDaylight SDN controller [7] for network management.
- Orchestration level is implemented in TeNOR Orchestrator module that includes VNF catalogue.

Interpretation of management and orchestration stack is combined from these two modules. Network connections between virtual infrastructure domains are created with SDN controller which manages edge gateways of virtual infrastructures simultaneously with VIMs.

One more project providing distributed cloud networking is BEACON [8]. Fig.4 depicts referenced architecture of BEACON with following components:

- BEACON Network manager enables inter-domain connections. It provides API for virtual networks creation independently of controlled domains and topology.
- BEACON network agent manages control plane for platform's network. It sends data about its network area to other network agents.
- BEACON datapath manages data plane using OpenFlow [9] to set rules. It defines traffic flows according to network agents' requests. It creates connections between domains and uses tunneling mechanisms to encapsulate traffic.
- SFC initiation and configuration are handled with BEACON network manager. This component also executes additional orchestration to coordinate all components.

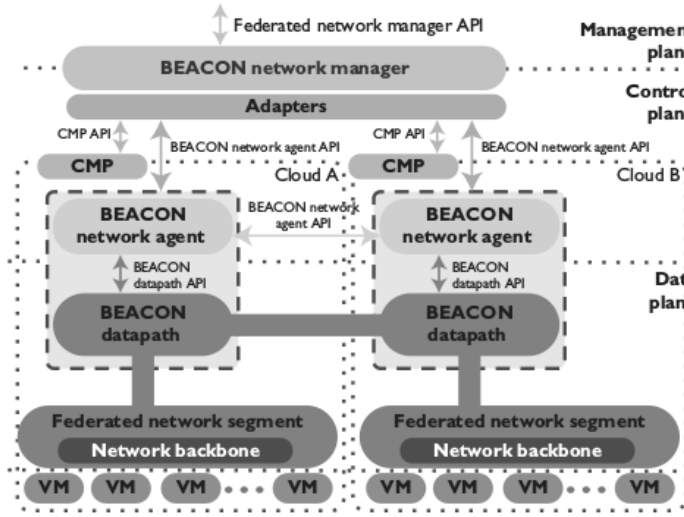

Fig. 4 BEACON architecture

We conclude this Section with following architecture features:

- WIM establishes inter-domain connections with SND controller,
- WIM uses tunneling mechanisms for inter-domain traffic,
- WIM shares virtual infrastructures' edge gateways control with virtual infrastructures' VIMs.

2. Designed System

In this Section the designed architecture based on ETSI management and orchestration reference architecture is proposed, considering conclusions from previous Section and additional requirements described further.

2.1 ETSI management and orchestration

WIM is functional block that manages inter-domain network connections with following functions [1]:

- WAN resources management and accounting,
- Providing interfaces to higher-layer modules (NFVO) for inter-domain connections instantiation,
- Providing QoS (Quality of Service) parameters such as bandwidth, RTT and latency.

NFVO-WIM interface should allow NFVO to request creation/deletion of inter-domain connection with required QoS parameters.

2.2 Additional requirements

WAN should be presented as virtual resource to manage it the same way as other platforms' resources [10]. This level of abstraction allows dynamically creating virtual connections; that is necessary for SFC provisioning. Virtual networks use overlay hypervisor networks thus making network between virtual machines owned by different services independent and secure and able to migrate with virtual machines it connects.

Virtual network consist of virtual links. Virtual link has bandwidth parameter which is shared by multiple connections (similar to storage size shared by multiple virtual machines) and some QoS parameters like RTT or latency. WIM manages network between virtual infrastructure domains and it should be able to measure bandwidth between domains to share it between SFCs deployed across WAN. To provide inter-VNF connections for different SFCs WIM should limit bandwidth for every connection to be sure that different SFCs does not affect each other. As WIM manages WAN it should be prepared for unexpected bandwidth decrease or other network characteristics change. For this reason WIM

should monitor controlled WAN characteristics and inform NFVO if critical changes happen.

Virtual resources are usually broken up into three groups: compute, storage and network. WAN segment may also be represented as virtual network but its characteristics change over time, that's why it becomes hardly predictable. Therefore we can define new type of virtual resources — dynamic virtual network. The virtual resources scheduler algorithm should be changed to consider that dynamic network state may be measured only in discrete moments of time. It also may need additional resource reservation and monitoring to provide the stable virtual infrastructure.

SDN technology enables separation of route selection and connection monitoring tasks that are a part of platform's logic and data transporting tasks that fulfilled by traditional networking hardware. This way SDN controller has a full control over inter-domain connections that leads to high scalability and flexibility in route configuration and provide dynamic automatic network reconfiguration. Consequently WIM can use SDN controller to manage WAN.

2.3 Designed Architecture

We specified ETSI architecture (Fig. 1) as shown in Fig. 5 taking into account described requirement and features. New element in this architecture is virtual infrastructure edge gateway that is endpoint for southbound WIM interface and also has interface for appropriate VIM.

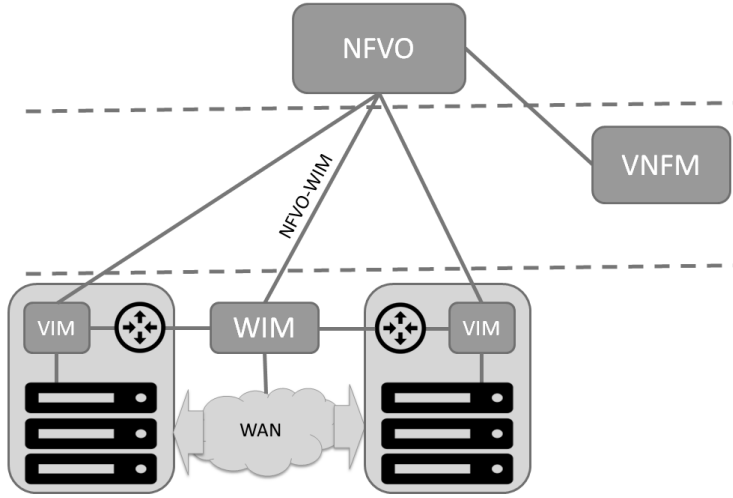

Fig. 5 Designed architecture

NFVO-WIM interface was also specified and should fulfill following requirements:

- WIM send information about current WAN characteristics (we suppose WAN has only bandwidth characteristic, QoS parameters are bypassed),
- NFVO send requests for creation/deletion if inter-domain connections specifying virtual infrastructure domains and edge gateway port,
- WIM notifies NFVO about created connection's characteristic interruption because of possible inter-domain link changes .

WIM functions were expanded as follows:

- WIM creates connections using tunneling technology,
- WIM allows setting bandwidth for connection,
- WIM notifies NFVO in case of connection bandwidth shortage.

2.4 Implementation

Developed system is based on single-domain NFV platform C2 [11] that was updated according to described architecture. Modules in C2 communicate through message queue RabbitMQ [12] therefore WIM uses it to implement interface to NFVO. When new VIM connects to platform it should configure its edge gateway and send needed information to control this gateway to WIM through NFVO.

We chose SDN controller OpenDaylight because it's open-source and support all needed functions described later in this subsection. As SDN-based gateways we use Open vSwitsh (OVS) [13] as VIMs use OpenStack virtual networks based on OVS and it supports OpenFlow 1.3 protocol [9] to communicate with SDN controller. We also use OVSDB plugin [14] in OpenDaylight to perform control operations with gateways that are not supported with OpenFlow.

WIM uses VXLAN as tunneling mechanism for every SFC and creates it with OpenDaylight. WIM set bandwidth restrictions for all SFC connections using OVS policing [15]. To monitor unused bandwidth of inter-domain links WIM uses queue prioritization. It creates two priority queues on every gateway queue with higher priority reserved for SFC traffic and the second queue is used for monitoring traffic. Monitoring traffic should not affect SFC traffic which means that queue with higher priority should have all available bandwidth and queue with

lower priority get the rest of bandwidth. Appropriate queuing policy implemented in OVS and is called linux-htb (Linux hierarchy token bucket) [16].

3. Experiments

This Section describes experiments performed to prove developed system usability. In NFV platforms managing single virtual infrastructure two types of SFC's VNFs placement exist: they can be placed on same compute node or on different compute nodes. In multi-domain NFV platforms VNFs also can be placed in different domains. Latency between VNFs deployed on different compute nodes composed of internal compute node latencies and additional inter-compute node latency. Also VNFs deployed in different domains experience higher latency between them than VNFs deployed in same domain. This latency composed of internal virtual infrastructure links latency, inter-virtual infrastructure WAN link latency and additional latency caused by traffic operations executed on edge gateways to enable inter-domain communication. Therefore we measured RTT for same SFC composed from two VNFs for all types of VNFs placement.

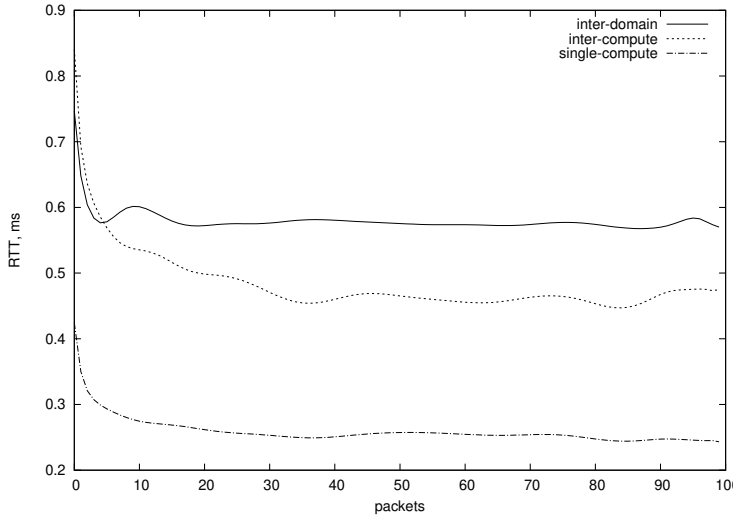

Fig. 6 RTT measurements for different SFC nodes placement types

Fig. 6 depicts experiment results. Measured values depend from number of packets sent in a row, for this reason x-axis shows packets amount and y-axis shows averaged measurement results. Latency for

VNFs deployed on same compute node is about 0.2 ms less than latency between VNFs placed in different compute nodes. To neutralize inter-domain latency we set its value close to inter-compute node latency that equal to 0.219 ms and 0.194 ms accordingly. Consequently, we can conclude that additional latency caused by inter-domain traffic processing is about 0.1 ms.

The next experiment is about inter-domain link bandwidth. As known, tunneling mechanisms cause some overheads. Therefore we measured dependency between SFCs amount deployed over inter-domain link and overall bandwidth used by SFCs' nodes. We shared bandwidth between SFCs equally and used iperf tool for every SFC to measure bandwidth. We made 200 tests, each of them has corresponding amount of SFC connections marked with VXLAN tag. We also measured inter-domain link bandwidth without any VXLAN and it is equal to 942 Mbps. Fig. 7 shows resulting curve and two lines indicating results for 100 and 200 VXLANs.

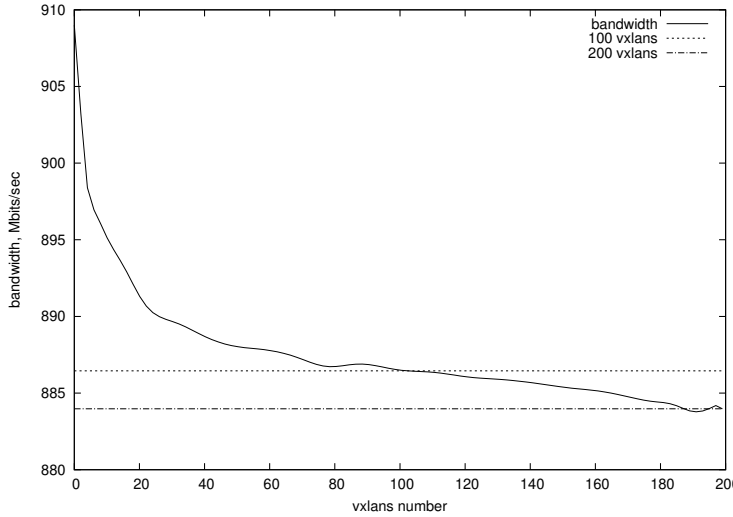

Fig. 7 Dependency curve for VXLANs amount and overall bandwidth

When using 1 VXLAN tunnel for 1 SFC we have 909 Mbps of bandwidth which is 3,5% less than without any VXLANs. With growth of VXLAN tunnels for SFC connections, overall bandwidth decreases slowly. Therefore overall bandwidth is equal to 886,455 Mbps with 100 VXLANs or 97,51% of 1 VXLAN bandwidth and 883,9745 Mbps with 200 VXLANs or 97,2% of 1 VXLAN bandwidth.

4. Conclusion

Distributed NFV platforms bring inter-domain networking challenges. We presented architecture providing inter-domain connections and its implementation. It enables creation and deletion of inter-domain connections with configurable bandwidth. It also has monitoring mechanism and therefore it implements control over WAN.

We defined inter-domain WAN segment as new virtual resource type — dynamic virtual network that requires some changes in scheduling algorithms considering its properties.

Experiments results shown that inter-domain connections have about 0.1 ms additional latency for traffic processing on edge gateways. We also demonstrated that SFCs amount has low influence on inter-domain link bandwidth. Therefore we discussed restrictions and additional overheads for inter-domain connections in NFV platform.

Distributed NFV platforms may be used to organize Edge Computing systems that are very popular now as a part of 5G technology. It also may be used for services that need ultra-low latency and high bandwidth like Internet of Things.

References

1. ETSI NFV ISG, *V1.1.1 Network Function Virtualisation (NFV); Management and Orchestration*, Dec. 2014.

2. ETSI NFV ISG, *V1.2.1 Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV. Group Specification*, Dec. 2014.

3. J. Halpern and C. Pignataro, *Service Function Chaining (SFC) Architecture*, Oct. 2015. [Online]. Available: `http://www.rfc-editor.org/rfc/rfc7665.txt`.

4. S. Draxler et al. *SONATA: Service programming and orchestration for virtualized software networks*, IEEE International Conference on Communications Workshops (ICC Workshops), pp. 973–978, May 2017.

5. M. A. Kourtis et al. *T-NOVA: An Open-Source MANO Stack for NFV Infrastructures*, IEEE Transactions on Network and Service Management, pp.586–602, Sept. 2017.

6. OpenStack, *OpenStack Docs: OpenStack API Documentation*, 2018. [Online]. Available: `https://developer.openstack.org/api-guide/quick-start/`.

7. J. Medved, R. Varga, A. Tkacik and K. Gray, *OpenDaylight: Towards a model-driven SDN controller architecture*, Proc. IEEE 15th Int. Symp. World Wireless Mobile Multimedia Netw., pp. 1–6, July 2014.

8. E. Huedo, R. S. Montero, R. Moreno, I. M. Llorente, A. Levin and P. Massonet, *Interoperable federated cloud networking*, IEEE Internet Computing, pp. 54–59, Sept. 2017.

9. B. Ptaff et al. *OpenFlow Switch Specification*, 2013. [Online]. Available: `https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf`.

10. A. Castiglione, F. Palmieri and K. K. R. Choo, *Enhanced network support for federated cloud infrastructures*, IEEE Cloud Computing, pp. 16–23, June 216.

11. V. Antonenko, R. Smeliansky, A. Ermilov, A. Plakunov, N. Pinaeva and A. Romanov, *C2: General Purpose Cloud Platform with NFV Life-Cycle Management*, Cloud Computing Technology and Science (CloudCom), 2017 IEEE International Conference, pp. 353–356, Dec. 2017.

12. RabbitMQ, *RabbitMQ - Messaging that just works*, 2017. [Online]. Available: `https://www.rabbitmq.com/`.

13. B. Pfaff et al., *The design and implementation of open vSwitch*, Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI), pp. 117–130, 2015.

14. OpenDaylight, *OVSDB Integration:Design - OpenDaylight Project*, 2015. [Online]. Available: `https://wiki.opendaylight.org/view/OVSDB_Integration:Design`. [Accessed July 20, 2018]

15. Openvswitch, Linux Foundation, *Quality of Service (QoS) — Open vSwitch 2.9.90 documentation*, 2018. [Online]. Available: `http://docs.openvswitch.org/en/latest/faq/qos/`.

16. D. G. Balan and D. A. Potorac, *Linux HTB Queuing Discipline Implementations*, NDT '09. First International Conference, July. 2009.

Васильева Ю.О., Костенко В.А., Морозов В.В.

Планирование вычислений в территориально распределенном ЦОД

Введение

Эффективность эксплуатации ресурсов центра обработки данных определяется используемыми алгоритмами планирования. В статье исследуется тема метапланирования над совокупностью локальных центров обработки данных. Подобный распределенный ЦОД может встречаться как у компаний, занимающихся предоставлением вычислительных ресурсов пользователям (например, «Ростелеком»), так и у корпораций, использующих ЦОД для обеспечения собственных потребностей в вычислениях (например, «Сбербанк»). Из заявлений представителей таких компаний, как Facebook, можно заключить, что ЦОД, предназначенный для внутренних нужд корпорации, обслуживает по большей части независимые задачи [1]. В связи с этим планирование вычислений крайне актуально и для корпоративных ЦОД.

1. Задача метапланирования над локальными ЦОД

Метапланировщик представляет собой отдельный этап механизма назначения пользовательских заявок на ресурсы центров обработки данных. Пользовательские заявки поступают в виде запросов, включающих в себя требование на выделение вычислительных ядер процессора и оперативной памяти (в виде виртуальных машин), пространства на жестких дисках (в виде виртуальных хранилищ данных). Помимо этого заявки содержат требования по пропускной способности – виртуальные каналы между виртуальными машинами и хранилищами.

Независимые ЦОД собраны в группу с целью обработки единого пула запросов. Схематично роль метапланировщика иллюстрирует рисунок 1.

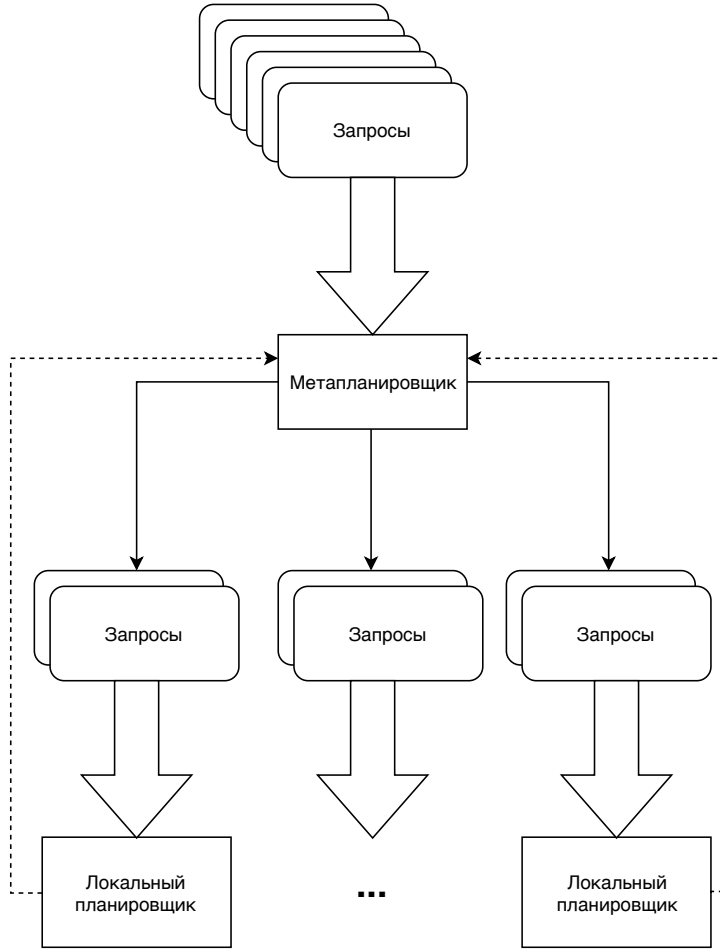

Рис. 1. Метапланирование над локальными планировщиками

Назначение локальным планировщиком запросов на ресурсы может быть реализовано, например, в виде жадной стратегии [6]. В подобной реализации производится попытка назначить на выполнение в первую очередь «тяжелые» запросы, то есть такие, которые превосходят остальные в смысле некоторой вводимой метрики.

Важно заметить, что планировщики локальных ЦОД между собой никак не взаимодействуют. Распределение запросов общего пула между локальными планировщиками реализуется исключительно на уровне метапланировщика.

Метапланировщик в рамках своего алгоритма отправляет локальным планировщикам наборы запросов на размещение. В своем

ответе локальные планировщики сообщают, какие запросы из набора оказываются размещенными, а какие нет. В расширенном варианте алгоритма указывается, что для размещения некоторых запросов можно снять уже исполняющиеся менее тяжелые запросы.

Процесс рассылки наборов и получения обратной связи является итеративным. Общий пул очередной итерации формируется из запросов, не размещенных на предыдущей итерации.

Цель метапланировщика – распределить запросы наиболее эффективным образом. Эффективность подразумевает следующее:

- размещение такого подмножества запросов, которое максимизирует загрузку вычислителей ЦОД;
- распределение запросов по локальным ЦОД наиболее равномерным образом;
- наименьшее число итераций в рамках заданного ограничения.

Вышеназванное продиктовано экономическими соображениями. Максимизация загрузки увеличивает доход провайдера ресурсов. Равномерное распределение запросов позволяет сократить издержки, которые в ЦОД растут нелинейно [4, 5]. Малое число итераций обеспечивает приемлемое время метапланирования.

2. Проблема планирования вычислений в распределенных системах

Помимо центров обработки данных, метапланирование находит место в таких системах, как GRID и облачные вычисления. Если в первом случае работа метапланировщика полностью автоматизирована и часто осуществляется на отдельном мастер-узле, то во втором уровень метапланирования возникает в виде статического плана, организуемого пользователем с целью распределить выполнение задачи на независимых виртуальных сущностях, которые можно разместить в том числе на ресурсах различных облачных хостинг-провайдеров.

Подходы, применяемые при планировании в GRID [2], учитывают следующие свойства вычислителей:

- гетерогенность по характеристикам;
- непостоянный состав вычислителей – новые узлы могут как подключаться к GRID, так и выбывать.

Метапланировщики GRID проектируются с учетом приведенных выше свойств, что означает дополнительные затраты при планировании: учитывается создание контрольных точек, миграция задач в связи с выбыванием вычислителя. Подобные затраты не возникают в распределенном ЦОД.

Пользовательское метапланирование [3] для облачных платформ осуществляется вне учета ограниченности ресурсов последних, так как ресурсные запросы отдельного пользователя почти всегда несопоставимо меньше физических ресурсов в распоряжении провайдера.

В ситуации с распределенным ЦОД совокупность запросов не всегда может быть назначена на исполнение в локальном ЦОД. В подобном случае важно предусматривать возможность обратной связи со стороны локальных планировщиков. Механизмы миграции задач с одной платформы на другую, аналогичные миграции запросов из одного локального ЦОД в другой, также остаются недоступными.

Метапланирование в контексте ЦОД обладает рядом особенностей, которые не позволяют перенимать подходы GRID и облаков. При формировании плана в GRID учитываются издержки, которые в случае с ЦОД не возникают. Пользовательский план облачных вычислений строится в предположении неограниченности доступных вычислительных ресурсов, в связи с чем такие вопросы, как перенаправление запросов от одного ЦОД к другому, на уровне метапланирования не решаются. Однако организация миграции запросов других пользователей ради размещения некоторого объемного запроса может найти место при метапланировании в рамках группы ЦОД.

Таким образом, проектирование алгоритма метапланирования в ЦОД должно осуществляться на основе принципов, отличных от принципов GRID и идей, закладываемых в пользовательское метапланирование для облачных платформ.

3. Подходы к построению алгоритма

3.1 Схемы метапланирования в ЦОД

Процесс назначения запросов общего пула локальным планировщикам может производиться последовательно или параллельно. В рамках последовательной схемы метапланировщик формирует непересекающиеся наборы запросов и направляет их локальным планировщикам. В параллельной схеме один и тот же запрос может попасть в наборы разных локальных планировщиков. По результатам обратной связи метапланировщик решает, какой ЦОД выбрать для

размещения подобного запроса.

В общем случае суммарные ресурсные требования могут превосходить свободные ресурсы распределенного ЦОД. В связи с этим важно определить, какие запросы будут попадать в наборы планировщиков в первую очередь. Выбор запроса может быть как рандомизированным, так и жадным, то есть приоритет в размещении отдается тяжелым запросам.

Также важно задавать стратегию определения локального ЦОД, в чей набор будет включен очередной в порядке приоритета запрос. Метапланировщик может выбирать ЦОД рандомизированно, а может опираться на имеющуюся информацию о ресурсах локальных ЦОД. Такой информацией может являться, например, сумма свободных ресурсов в ЦОД или же граф остаточных ресурсов, то есть свободные ресурсы отдельных серверов и хранилищ этого ЦОД.

Приведенные выше варианты выбора запросов и ЦОД сравниваются между собой в контексте утилизации ресурсов и возможности размещать большее число тяжелых запросов.

Ситуацию, в которой локальный планировщик принимает тяжёлый запрос при условии снятия менее ресурсоемких, можно рассматривать как новую задачу метапланирования. В этой задаче набор запросов состоит из снимаемых с локального ЦОД. Таким образом, имеет место рекурсивный вызов алгоритма метапланирования, при этом внутренняя подзадача предполагает размещение именно всех снимаемых запросов. Задавать ограничение на глубину рекурсии следует из тех соображений, что большое число рекурсивных вызовов замедляет процесс метапланирования, а миграция выполняемых запросов из одного ЦОД в другой может обходиться весьма дорого.

3.2 Алгоритм метапланирования в последовательной схеме

Алгоритм работы метапланировщика, отрабатывая за конечное число итераций, распределяет запросы общего пула по локальным планировщикам. Часть запросов пула могут остаться не определены ни одному локальному планировщику. На вход алгоритм получает ограничение на число итераций обращений к локальным планировщикам, описание локальных ЦОД, а также множество запросов общего пула. Заранее задаются стратегия выбора очередного запроса и стратегия выбора ЦОД, кандидатом в набор которого будет рассмотрен очередной запрос. Основной алгоритм выглядит следующим образом.

1. Инициализация. Завести счётчик итераций. Сопоставить каж-

дому запросу общего пула переменную размещения, в которой будет храниться идентификатор локального ЦОД, в котором этот запрос размещен, или пустое значение, если имеющаяся информация о ЦОД позволяет сделать вывод о том, что разместить запрос невозможно ни на одном ЦОД. Положить значение переменной неопределенным.

2. Увеличить счётчик числа итераций на единицу. Если его значение превысило максимальное число итераций, перейти на шаг 11.

3. Сопоставить каждому ЦОД множество, в котором будет формироваться набор запросов для отправки локальным планировщикам в рамках итерации. Положить эти множества пустыми.

4. Сформировать множество G из запросов, у которых значение переменной размещения не определено.

5. Если множество G пусто, перейти на шаг 8, иначе, руководствуясь стратегией выбора запроса, определить запрос g из множества G, исключить его из G.

6. В соответствии с имеющейся информацией о ЦОД определить множество D тех ЦОД, в которых представляется возможным разместить запрос g.

7. Если множество D пусто, установить в переменную размещения запроса g пустое значение и перейти на шаг 5. Иначе, руководствуясь стратегией выбора ЦОД, определить ЦОД d из множества D и внести запрос g в сопоставленное d множество-набор запросов итерации. Учесть в имеющейся о ЦОД d информации факт того, что запрос g включён в его набор. Последнее производится с целью в дальнейшем не включать в этот набор запросы, которые заведомо не удастся разместить в случае удачного размещения g. Перейти на шаг 5.

8. Отослать локальному планировщику каждого ЦОД соответствующее этому ЦОД множество-набор запросов итерации. По результатам обратной связи удачно размещенным запросам установить в переменную размещения соответствующий идентификатор ЦОД. Зафиксировать новую имеющуюся о ЦОД информацию.

9. В расширенном варианте алгоритма обратная связь также содержит данные о том, какие запросы можно разместить за счёт

снятия менее тяжелых. Для таких наборов менее тяжелых запросов последовательно рекурсивно запустить алгоритм метапланирования, где в качестве максимального числа итераций будет выступать текущее значение счетчика итераций, в качестве набора запросов – данный набор-условие, а информация о ЦОД представляется таким образом, будто соответствующий тяжелый запрос уже размещен. Если в такой ситуации набор-условие размещается полностью, считать соответствующий набору тяжелый запрос размещенным. Зафиксировать новую информацию о ЦОД.

10. Перейти на шаг 2.

11. Алгоритм метапланирования завершен. Переменные размещения, соответствующие запросам общего пула, хранят данные о ЦОД, в которых эти запросы размещены.

3.3 Принцип работы локального планировщика

В качестве примера локального планировщика рассмотрим алгоритм, сочетающий в себе жадную стратегию и ограниченный перебор [6]. Алгоритм предполагает назначение виртуальных машин и хранилищ данных на физические сущности, а затем прокладывание маршрутов в сети ЦОД, соответствующих виртуальным каналам.

Процесс отображения виртуальных сущностей на физические включает в себя следующие шаги:

1. если множество запросов не пусто, в соответствии с заданным жадным критерием для запросов выбирается очередной запрос, иначе алгоритм считается завершенным;

2. если запрос имеет не назначенные на физические ресурсы элементы, то, руководствуясь жадным критерием для виртуальных элементов, выбирается очередной элемент, иначе осуществляется переход на шаг 1;

3. для выбранного на предыдущем шаге виртуального элемента формируется множество физических ресурсов, на которых этот элемент может быть размещен, и в случае непустого множества назначение производится в соответствии с жадным критерием для ресурсов, производится переход на шаг 2. Иначе происходит запуск процедуры ограниченного перебора, цель которой – скомпоновать виртуальные сущности на физических ресурсах таким образом, чтобы появилась возможность разместить

выбранный элемент. Если результат подобной процедуры оказывается неуспешен, считается, что запрос с данным элементом не может быть назначен, и все сформированные ранее назначения элементов запроса на физические ресурсы удаляются. Следует переход на шаг 1.

Процедура ограниченного перебора, помимо переназначений элементов запросов в рамках одного ЦОД, может предполагать возможность обращения к метапланировщику с предложением о снятии с себя нетяжелых запросов. В этом случае по результатам планирования метапланировщику помимо построенного плана возвращается список снимаемых запросов. Если метапланировщик способен назначить данные запросы на другие ЦОД, план считается принятым, иначе он перестраивается.

4. Экспериментальные исследования

При экспериментальном исследовании последовательной схемы сравниваются такие показатели метапланирования, как число размещенных запросов, количество итераций алгоритма, число утилизированных процессорных ядер. Первый показатель дает возможность оценить, насколько хорошо алгоритм размещает запросы в среднем, второй показатель характеризует эффективность производимых итераций, третье значение позволяет понять, как обстоят дела с размещением тяжелых запросов.

При исследовании на генерируемых данных алгоритм метапланирования запускается 100 раз, после чего результаты запусков агрегируются: суммируется число размещенных запросов, число проделанных итераций, число утилизированных ядер процессора.

В результате проведения экспериментов не удалось установить вариант алгоритма, который превосходил бы все остальные по эффективности в контексте всех трех изучаемых параметров.

В процессе проверки гипотез и работы с генерируемыми данными подтвержден или выявлен ряд особенностей. Ниже приведены некоторые из них.

1. Алгоритм в случае детализированного описания ЦОД осуществляет метапланирование за меньшее число итераций, почти всегда хватает одной итерации. Для описания в виде совокупности ресурсов число итераций иногда в среднем может достигать 3-х. Очевидно, сказывается меньшая осведомленность алгоритма метапланирования.

2. Большое число итераций для алгоритма с суммарной оценкой ресурсов появляется в случае жадных стратегий выбора. Это означает, что ограничение в таких случаях числа итераций меньшим значением, например, 2, сильно скажется на эффективности алгоритма. Таким образом, в случае использования представления в виде суммарной оценки ресурсов, необходимо обеспечивать достаточное число итераций, иначе алгоритм деградирует.

3. Применение в обоих случаях рандомизированной стратегии выбора элемента увеличивает число обслуженных запросов в 1.6 раза относительно обеих жадных стратегий. Однако число утилизированных ядер снижается на 6%. Число итераций в случае детализированного представления растет почти незначительно.

4. Рандомизация только выбора запроса увеличивает число обслуженных запросов в 1.5 раза, рандомизация только выбора ЦОД почти не увеличивает. Очевидно, первое ведет к обслуживанию запросов, которые в среднем легче.

Заключение

В статье рассмотрены особенности организации метапланирования в распределенном ЦОД, произведено сопоставление с пользовательским метапланированием для облачных платформ и метапланировщиками GRID-систем.

В работе приведены различные схемы взаимодействия метапланировщика с локальными планировщиками, предложен алгоритм метапланировщика в последовательной схеме взаимодействия. Проведены экспериментальные исследования реализации последовательной схемы.

В результате экспериментального исследования выявлены принципы, позволяющие подстраивать алгоритм метапланирования под определенные нужды.

Литература

1. Long Thai, Blesson Varghese, Adam Barker *A Survey and Taxonomy of Resource Optimisation for Executing Bag-of-Task Applications on Public Clouds* // Future Generation Computer Systems, Vol. 82, 2018, pp. 1 - 11.

2. Прихожий А. А., Фролов О. М. *Исследование планировщиков задач в GRID* // Системный анализ и прикладная информатика: международный научно-технический журнал, 2015, № 1, с. 15 - 23.

3. Marco A. S. Netto, Rajkumar Buyya *Coordinated Rescheduling of Bag-of-Tasks for Executions on Multiple Resource Providers* // Concurrency and Computation: Practice & Experience, 2012, Vol. 24, Issue 12, pp. 1362 - 1376.

4. Miyuru Dayarathna, Yonggang Wen, Rui Fan *Data Center Energy Consumption Modeling: A Survey* // IEEE Communications Surveys & Tutorials, 2016, Vol. 18, Issue 1, pp. 732 - 794.

5. Anshul Gandhi, Rajarshi Das, Mor Harchol-Balter, Charles Lefurgy *Optimal Power Allocation in Server Farms* // Proc. 11th SIGMETRICS Int. Joint Conf. Meas. Model. Comput. Syst., 2009, pp. 157 - 168.

6. 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.

Волканов Д.Ю., Малышев Н.С.

Оптимизация надёжности распределённых вычислительных систем с модульной архитектурой[1]

Введение

Распределённые вычислительные системы (РВС) представляют собой набор независимых вычислитей, видимый пользователем в качестве единого целого и направленный на решение определённого спектра задач [1].

Надёжность распределённой вычислительной системы - это способность РВС сохранять работоспособность при определённых спецификацией условиях в течении заданного периода времени. Для оценки надёжности используют каким-либо образом описанную вероятность R безотказной работы системы в течении заданного интервала времени при том условии, что система работоспособна в начальный момент времени [1].

В данной статье рассматриваются системы с модульной архитектурой, состоящие из универсальных модулей, каждый из которых способен выполнять подмножество задач из некоторого заданного набора. Общие свойства и подходы к описанию и оптимизации надёжности РВС верны и для рассматриваемых систем, но в данном случае задачи могут быть динамически перераспределены между модулями при необходимости. Динамическое перераспределение нагрузки между модулями системы будем называть реконфигурацией системы.

Рассматривается такой режим работы, при котором функционируют все аппаратные и программные составляющие системы, то есть одним из условий работоспособности является отсутствие аппаратных и программных отказов, кроме тех, которые компенсируются реконфигурацией, в течение всего заданного интервала времени.

Повышение надёжности достижимо при помощи следующих способов:

- Использование более надёжных модулей при конструировании системы;
- Применение методов обеспечения отказоустойчивости (МОО) к модулям системы.

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

Использование любого способа приводит к увеличению стоимости. Актуальной задачей является поиск такого набора компонент системы и МОО, что система обладает лучшей возможной надёжностью и удовлетворяет некоторым ограничениям, наложенным на стоимость всей системы. Данная задача относится к классу задач, называемых в литературе задачами оптимизации надёжности вычислительной системы.

В большинстве существующих исследований, что показано в работах [1,2,3,4,5,6], рассматривается исключительно надёжность с возможностью аппаратного резервирования, используется детерминистический подход к описанию характеристик системы, в том числе надёжности, а надёжности аппаратных и программных компонентов модулей рассматриваются отдельно.

Для решения поставленной задачи был выбран более реалистичный, что отмечено в работах [2,3,4,5,6], способ описания надёжности - в виде вещественных интервалов, выбран больший спектр МОО, а также проведён обзор работ, решающих близкие к поставленной задачи.

1. Методы обеспечения отказоустойчивости

Целью обзора МОО в работе [1] являлось формирование набора доступных МОО для последовательно-параллельных распределённых вычислительных систем. По результатам обзора были выбраны следующие МОО:

- Активное резервирование (AR/1/0);
- N-версионное программирование (NVP/0/1);
- N-версионное программирование вкупе с активным резервированием (NVP/1/1);
- Восстановление блоками вкупе с активным резервированием (RB/1/1);

Формулы расчёта надёжности и стоимости, описанные в работе [1], применимы и для интервальной надёжности, но с учётом специфики арифметических операций с интервалами.

Кроме этих МОО, РВС с модульной архитектурой позволяет использовать динамические перераспределение вычислительной нагрузки между модулями. Предлагается рассматривать возможность

реконфигурации, как новое МОО: *"аппаратная реконфигурация с возможностью двух последовательных отказов"(HWRC/2/0)*. При возможности реконфигурации отказ модуля наступит только в том случае, если после первичного отказа аппаратной компоненты модуля U_i, откажет не только аппаратная составляющая модуля U_j, на который была перенесена вычислительная нагрузка, но и аппаратная часть модуля U_k, на который в свою очередь была перенесена нагрузка после вторичного отказа. При переносе нагрузки с отказавшего модуля на другой модуль, на последнем выполняется не только перенесённая, но и его собственная программная компонента. Опишем формулы для вычисления надёжности модуля с HWRC/2/0.

Обозначим как $Z(i)$ множество модулей U_j, на которые может быть перенесена вычислительная нагрузка с модуля U_i после первого отказа, а $Z(i,j)$ - множество доступных для переноса с U_j модулей U_k после второго отказа. Будем считать, что вероятность выбора модуля из множества Z при реконфигурации равна $1/|Z|$. Обозначим также как R_d и P_d надёжность и вероятность безотказной работы голосователя для NVP/0/1 и NVP/1/1, контрольного теста для RB/1/1 или контроллера при реконфигурации. $P_d = 1 - R_d$. Тогда надёжность R_i модуля U_i можно вычислить следующим образом:

$P_{HWRC-2}(i,j) = \sum_{k:U_k \in Z(i,j)} (1/|Z(i,j)| * P_k^H)$

$P_{HWRC-1}(i) = \sum_{j:U_j \in Z(i)} (1/|Z(i)| * P_j^H * (P_d + R_d * P_{HWRC-2}(i,j)))$

$R_i = R_i^S * (1 - P_i^H * (P_d + R_d * P_{HWRC-1}(i)))$

2. Постановка задачи оптимизации надёжности

Системы с модульной архитектурой состоят из n модулей, каждый из которых содержит не менее, чем одну версию как программного так и аппаратного компонентов этого модуля. Количество версий компонентов зависит от используемого метода обеспечения отказоустойчивости. Для каждого модуля определены набор версий программных и аппаратных компонент, которые могут быть использованы. Каждая программная и аппаратная компонента характеризуется своими надёжностью R_i^{type}, выраженной вещественным интервалом, и стоимостью C_i^{type}, выраженной целым числом, где $type = \{H, S\}$.

Конфигурация модуля U_i - описание модуля U_i в виде набора $< H_i, S_i, F_i >$ списков выбранных аппаратных H_i и программных S_i компонентов, а также используемого МОО F_i (может быть указано, что никакой МОО не используется). Конфигурация системы - набор конфигураций модулей $\{< H_i, S_i, F_i > | i = 1..n\}$, где n - количество

модулей в системе.

Задача состоит в нахождении конфигурации системы с модульной архитектурой с максимальной надёжностью при некоторых ограничениях на стоимость всей системы.

Надёжность R_{system} и стоимость C_{system} всей системы вычисляются по следующим формулам:

$$R_{system} = \prod_{i=1}^{n} R_i$$

$$C_{system} = \sum_{i=1}^{n} C_i$$

где n - число модулей в системе, R_i - надёжность i-го модуля, C_i - его стоимость. Надёжность и стоимость каждого модуля зависят от используемого МОО.

Пусть $systems$ - множество всех возможных конфигураций системы, $result$ - результирующая конфигурация, R_{result} и C_{result} - её надёжность и стоимость. Решаемую задачу можно сформулировать следующим образом - необходимо определить такую конфигурацию системы $result \in systems$, что:

$$\begin{cases} R_{result} = \max_{systems} R_{system}, \\ C_{result} \leq C_{system}^{max} \end{cases}$$

3. Существующие методы решения

Был проведён обзор существующих решений задачи оптимизации надёжности РВС, которые возможно расширить до решения задачи для систем с модульной архитектурой и интервальными оценками надёжности.

Опишем критерии, согласно которым производился выбор алгоритма решения поставленной задачи, и приведём используемые в таблице сокращения:

- Надёжность выражена интервалом (**ИН**);
- При оптимизации конфигурации системы учитывается специфика интервальной оценки надёжности (**ИС**);
- Аппаратная и программная отказоустойчивость рассматриваются совместно (**А+П**);
- Различные модули могут иметь различные с точки зрения надёжности и стоимости аппаратные и программные компоненты (**РМ-РК**);

- В различных конфигурациях модуль может иметь разные с точки зрения надёжности и стоимости аппаратные и программные компоненты (**РК-РК**);

- Множество МОО содержит набор, сформированный в данной работе (**МОО**).

Все рассматриваемые работы посвящены решению задачи оптимизации надёжности последовательно-параллельной РВС с интервальной надёжностью, кроме работы [1],в которой рассматривается детерминированная надёжность.

В работах [3] и [4] решается задача оптимизации надёжности исключительно аппаратной составляющей системы, состоящей из N модулей, в каждом из которых содержится один или более параллельно работающих идентичных аппаратных компонентов. В обоих иследованиях используется пессимистический подход к принятию решения и выборе наиболее надёжной конфигурации. Поставленная задача решается генетическим алгоритмом. При этом для каждого модуля определен аппаратный компонент и в процессе оптимизации изменяется только количество резервных компонентов в каждом модуле, то есть решается так называемая задача оптимального резервирования.

Работа [10] отличается от предыдущих тем, что в ней предлагается максимизировать функцию, зависящую от логарифмов левой и правой границ интервальных значений надёжности компонентов и вещественного параметра, принадлежащего отрезку [0, 1]. Однако максимизация этой целевой функции соответствует максимизации центра и левой/правой границы надёжности системы, выраженной интервалом. Параметр является весом, указывающим на приоритет левой/правой границы перед центром при вычислении целевой функции. Оптимизация производится при помощи генетического алгоритма и алгоритма имитации отжига.

В работе [1] в ходе оптимизации надёжности системы изменяется как набор программных и аппаратных компонентов модулей системы, так и используемые в них МОО, то есть решаемая задача относится к комплексным задачам оптимизации надёжности. При вычислении надёжности модулей совместно учитываются надёжности как аппаратных, так и программных компонентов. Единственным МОО, отсутствующим в данной работе, является аппаратная реконфигурация, что связано с неприменимостью данного МОО к РВС в общем случае. Однако, в данной работе рассматривается детерминистическая оценка надёжности компонентов модулей системы. Для решения стоящей в рамках рассматриваемой работы задачи были

применены классический и адаптивный эволюционные алгоритмы.

Наименование	ИН	ИС	А+П	РМ-РК	РК-РК	МОО
Bhunia A. K., Sahoo L., Roy D., 2010 [3]	+	+	-	+	-	-
Gupta R. K., Bhunia A. K., Roy D., 2009 [4]	+	+	-	+	-	-
Taguchi T., Yokota T., 2011 [10]	+	+	-	+	-	-
Волканов Д.Ю., 2016 [1]	-	-	+	+	+	±

Таблица 1. Обзор существующих решений.

Методы решения задач, поставленных в рамках рассмотренных работ, не подходят для решения задачи оптимизации надёжности системы с модульной архитектурой, возможностью реконфигурации и интервальными оценками надёжности компонент модулей. Наиболее близким по возможностям, является подход, применённый в работе [1]. Поэтому для решения поставленной задачи были разработаны модификации классического и адаптивного эволюционного алгоритма, предложенных в работе [1], учитывающие особенности интервального задания надёжности компонентов системы и возможности реконфигурации.

4. Эволюционный алгоритм и реализация

Как и в работе [1] алгоритмы решения поставленной задачи представляют собой классический и адаптивный эволюционный алгоритм. Результаты работы адаптивного алгоритма в каждом поколении анализируются, и, исходя из полученной информации, параметры алгоритма корректируются. Опишем шаги алгоритма.

1. **Кодирование решений**. Каждое решение, то есть конфигурация системы, кодируется в виде строки-хромосомы. Каждый модуль в конфигурации представляется геном хромосомы и представляет собой тройку <H, S, F>, где H и S - списки порядковых номеров аппаратных и программных компонентов из множества доступных в модуле, F - порядковый номер МОО из множества доступных в модуле. По каждому гену может

быть вычислена надёжность и стоимость модуля, соответствующего этому гену, и по каждой строке может быть вычислена надёжность и стоимость конфигурации системы, соответствующей этой строке.

2. **Начальная подготовка популяций решений**. Случайным образом генерируется популяция решений $systems_k$, $k \in [1, num]$, где k - число особей в начальной популяции, являющееся параметром алгоритма. Используемые МОО, а также аппаратные и программные составляющие случайной конфигурации выбираются согласно дискретному равномерному распределению. Переход к пункту 7 для оценки начальной популяции.

3. **Селекция**. Аналогично работе [1], используется схема селекции на основе заданной шкалы. Популяция сортируется, отбираются лучшие $X\%$ решений, которые затем участвуют в операции скрещивания. Сортировка решений осуществляется в соответствии с пессимистическим и оптимистическим подходами, как в работах [3,4,5,6], для левой и правой границы интервала надёжности, а также подходом, основанным на вычислении расстояния Мура, описанного в работах [7,8,9], до лучшего решения в популяции.

4. **Скрещивание**. Одноточечное скрещивание модулей осуществляется num раз, в результате чего появляется временная популяция из $2 * num$ решений.

5. **Формирование новой популяции**. Промежуточная популяция сортируется аналогично пункту 4, а затем уже в новую популяцию выбирается $Y\%$ лучших решений исходной популяции, и дополняется до количества num лучшими особями промежуточной популяции.

6. **Мутация**. Часть лучших решений не мутирует, к остальным с некоторой вероятностью применяется одноточечная мутация, то есть в решении случайным образом выбирается ген-модуль. Для этого модуля случайным образом меняются доступные для использования МОО и аппаратные и программные компоненты. Ген в решении меняется соответственно.

7. **Оценка популяции**. Проверяется ограничение стоимости $C_{system} < C^{max}_{system}$ и вычисление целевой функции R_{system} для каждого решения в популяции. Если решение не удовлетворяет

ограничениям, то вычисляется штрафная функция со значением в интервале $(0; 1)$, на значение которой умножается надёжность решения.

8. **Проверка критерия останова**. При достижении максимально допустимого числа итераций без изменения лучшего решения осуществляется переход к пункту 10, иначе - к пункту 9.

9. **Изменение значений параметров алгоритма**. Осуществляется подстройка параметров алгоритма на основании информации, полученной при анализе популяции при помощи блока автоматического изменения значений параметров алгоритма. Переход к пункту 3.

10. **Завершение работы**. В качестве результата выбирается лучшая конфигурация системы *result* из найденных.

Блок автоматического изменения значений параметров алгоритма описан в работе [1]. Основываясь на входных параметрах в виде среднего и лучшего значений надёжности текущей и предыдущей популяций происходит выбор соответствующего значения вероятности скрещивания, доли лучших особей популяции, которые не будут скрещиваться, вероятности мутации, доли лучших особей популяции, не подверженных мутации, а также степени мутации гена.

Для реализации была проведена интеграция с программным средством RelOpt [1]. В связи с интервальным характером надёжности были изменены механизмы ввода и вывода данных, формулы расчёта надёжности модулей и всей системы и реализации классического и адаптивного эволюционных алгоритмов. В частности были реализованы пессимистический, оптимистический и использующий расстояние Мура подходы к сравнению заданной интервалом надёжности двух решений и адаптации параметров алгоритма. Был добавлен класс, соответствующий модулю с возможностью динамического перераспределения вычислительной нагрузки.

5. Экспериментальное исследование

Были поставлены следующие цели исследования:

- Сравнение значения целевой функции при использовании классического (КЭА) и адаптивного (АЭА) эволюционых алгоритмов;

- Сравнение влияния набора доступных МОО на надёжность результата работы алгоритма.

Для каждого запуска алгоритма фиксировалась конфигурация системы, её надёжность и стоимость. Каждый алгоритм запускался 300 раз с популяцией размера 30.

Экспериментальная система имела параметры, соответствующие работе [1], но с интервальной надёжностью и следующим набором МОО: без МОО, NVP/0/1, NVP/1/1, RB/1/1, HWRC/2/0.

Алгоритм		КЭА	АЭА
C^{max}_{system}			
180	Левая граница	0.604	0.596
	Правая граница	0.801	0.785
320	Левая граница	0.731	0.723
	Правая граница	0.898	0.839
460	Левая граница	0.802	0.801
	Правая граница	0.932	0.935
∞	Левая граница	0.917	0.918
	Правая граница	0.972	0.972

Таблица 2. Средние значения левой и правой границ интервала надёжности.

АЭА демонстрирует худшие средние значения границ интервала при сильных ограничениях на стоимость и только при ослаблении этого ограничения начинает сближаться с КЭА.

C_{system}^{max}	Характеристика	HWRC/2/0	Без HWRC/2/0
240	Min левая граница	0.595	0.561
	Max левая граница	0.689	0.749
	Avg левая граница	0.601	0.655
	Min правая граница	0.653	0.587
	Max правая граница	0.809	0.787
	Avg правая граница	0.798	0.690
320	Min левая граница	0.595	0.567
	Max левая граница	0.595	0.801
	Avg левая граница	0.595	0.699
	Min правая граница	0.809	0.592
	Max правая граница	0.809	0.861
	Avg правая граница	0.809	0.747
∞	Min левая граница	0.595	0.800
	Max левая граница	0.595	0.893
	Avg левая граница	0.595	0.845
	Min правая граница	0.809	0.904
	Max правая граница	0.809	0.961
	Avg правая граница	0.809	0.939

Таблица 3. Сравнение аппаратной реконфигурации с другими МОО.

Очевидно, что для достижения улучшения результата работы АЭА по сравнению с КЭА, сравнимого с оригинальным алгоритмом из работы [1], необходимо использовать другие подходы к сравнению интервалов или более сложные механизмы адаптации параметров алгоритма, позволяющие учитывать больше характеристик интервала.

Аппаратная реконфигурация при сильных ограничениях на стоимость позволяет получить более высокие правую границу надёжности и минимум левой границы надёжности, чем использование других МОО. Превосходство других МОО при слабых или отсутствующих ограничениях на стоимость системы связано с тем, что рассматриваемый тип реконфигурации не улучшает надёжность программной составляющей модулей системы в отличии от N-версионного программирования и восстановления блоками. Надёжность программных компонентов обычно ниже надёжности аппаратных компонентов и при вычислении общей надёжности системы это приводит к значениям надёжности, уступающим системам, использующим МОО, повышающие надёжность программной составляющей.

Заключение

В данной статье рассмотрена задача оптимизации надёжности РВС с модульной архитектурой, приведены аргументы в пользу определения надёжности компонент модулей в интервальном виде, описан механизм работы нового аппаратной реконфигурации HWRC/2/0 как нового МОО, коротко описаны изменения, внесённые в механизм работы эволюционого алгоритма средства RelOpt.

Экспериментальное исследование показало недостаточность пессимистического, оптимического и использующего расстояние Мура подходов к сравнению интервалов при решении поставленной задачи оптимизации. Также исследование продемонстрировало эффективность аппаратной реконфигурации при сильных ограничениях, накладываемых на стоимость системы, и необходимость использования этого МОО вкупе с каким-либо программным МОО при ослаблении ограничений на стоимость.

Литература

1. Волканов Д. Ю. *Метод сбалансированного выбора механизмов обеспечения отказоустойчивости для распределённых вычислительных систем.* // Моделирование и анализ информационных систем. — 2016. — Т. 23, № 2. — С.119–136.

2. Soltani R. *Reliability optimization of binary state non-repairable systems: A state of the art survey.* // International Journal of Industrial Engineering Computations. – 2014. – Т. 5. – №. 3. – Pp. 339-364.

3. Bhunia A. K., Sahoo L., Roy D. *Reliability stochastic optimization for a series system with interval component reliability via genetic algorithm.* // Applied Mathematics and Computation. – 2010. – Т. 216. – №. 3. – Pp. 929-939.

4. Gupta R. K., Bhunia A. K., Roy D. *A GA based penalty function technique for solving constrained redundancy allocation problem of series system with interval valued reliability of components.* // Journal of Computational and Applied Mathematics. – 2009. – Т. 232. – №. 2. – Pp. 275-284.

5. Sahoo L., Bhunia A. K.,Roy D. *Reliability optimization with high and low level redundancies in interval environment via*

genetic algorithm. // International Journal of System Assurance Engineering and Management. – 2014. – Т. 5. – №. 4. – Pp. 513-523.

6. Sahoo L., Bhunia A. K., Roy D. *A genetic algorithm based reliability redundancy optimization for interval valued reliabilities of components.* // Journal of Applied Quantitative Methods. – 2010. – Т. 5. – №. 2.

7. Trindade R. M. P. et al. *An interval metric.* // New Advanced Technologies. – InTech, 2010.

8. Bohlender G., Kulisch U. *Definition of the arithmetic operations and comparison relations for an interval arithmetic standard.* // Reliable Computing. – 2011. – Т. 15. – P. 37.

9. Hickey T., Ju Q., Van Emden M. H. *Interval arithmetic: From principles to implementation.* // Journal of the ACM (JACM). – 2001. – Т. 48. – №. 5. – Pp. 1038-1068.

10. Taguchi T., Yokota T. *A solution method for nonlinear integer programming problem with interval coefficients using hybrid ga/sa algorithms.* // Proceedings of the 41st international conference on computers & industrial engineering. – 2011.

Ковалёв А.И., Степанов Е.П.

Разработка метода адаптивного выбора алгоритмов управления перегрузкой для транспортных соединений[1]

Аннотация

Для борьбы с явлением перегрузки в сети были разработаны многочисленные алгоритмы управления перегрузкой. Цель таких алгоритмов – сохранить наивысшую скорость соединения, при этом предотвратив потери пакетов из-за перегрузки. Разные алгоритмы разработаны для управления перегрузкой в сетевых соединениях с разными параметрами качества (пропускная способность, задержка, процент потерь и т.д.), и использование неподходящего алгоритма ведёт к значительному снижению скорости соединения. Несмотря на это, большинство распределённых приложений полагается на стандартные алгоритмы управления перегрузкой, так как разработчики этих приложений жертвуют производительностью ради простоты разработки. В данной работе предложен метод детектирования характеристик сетевого окружения и выбора оптимального алгоритма управления перегрузкой на основании этих характеристик.

1. Введение

Узлы современных компьютерных сетей подвержены явлению перегрузки. Данное явление возникает в случае, если скорость поступления пакетов на узел сети выше скорости, с которой этот узел может их отправлять, в течение некоторого достаточно длительного промежутка времени. При наступлении перегрузки узел сети начинает сбрасывать пакеты, что ведёт к значительному снижению скорости передачи трафика. Одним из средств борьбы с этим явлением являются алгоритмы управления перегрузкой, которые регулируют интенсивность отправки трафика в сеть. Эти алгоритмы используются рядом протоколов транспортного и прикладного уровней, в том числе широко распространённым протоколом TCP.

Алгоритмы управления перегрузкой обычно разрабатываются для определённых сетевых окружений: Westwood [9] и Veno [11] пред-

[1]Работа выполнена при частичной поддержке РФФИ, грант № 18-07-01255.

назначены для работы в беспроводных сетях, BBR [10], Cubic [6] и Compound [12] предназначены для проводных соединений с высокой пропускной способностью и высокой задержкой. Использование неподходящего алгоритма приводит к потерям производительности. Таким образом, в сетевых окружениях с разными характеристиками выгодно использовать подходящие под них алгоритмы управления перегрузкой, однако возможности выбора алгоритма управления перегрузкой на уровне операционной системы обычно сильно ограничены. Например, в операционных системах с ядром Linux можно лишь выбрать стандартный алгоритм управления перегрузкой, который будет использован для всех последующих TCP сессий, где алгоритм явно не задаётся.

Существуют разные подходы к оптимизации управления перегрузкой, включающие создание новых алгоритмов или манипуляции с уже существующими. Создание универсального алгоритма, превосходящего существующие при любых характеристиках качества соединения, маловероятно. Поэтому рассматривались подходы к оптимизации управления перегрузкой без создания нового алгоритма. В результате анализа научных работ были рассмотрены средства Remy [3], TCP Tuner [1] и AC/DC [2].

Remy - средство автоматической настройки алгоритмов управления перегрузкой для узлов сети на основании информации о сети(топология, пропускные способности, задержки и т.д.) и целевой функции. Настройка алгоритмов требует нескольких часов, поэтому данное средство не позволяет подстраиваться под изменения характеристик качества соединения.

TCP Tuner - средство для настройки параметров алгоритма управлений перегрузкой Cubic. Настройку можно проводить вручную при помощи графического интерфейса.

AC/DC - техника для имитации поведения алгоритма управления перегрузкой при помощи перехвата пакетов и изменения размера окна получателя. Было показано, что AC/DC позволяет успешно имитировать алгоритм DCTCP [4], повышаю утилизацию сети в центрах обработки данных.

Все рассмотренные средства использовали заранее заданные алгоритмы, что не оптимально в случае изменяющихся характеристик качества. Кроме того, Remy требовал предварительных знаний о сети, TCP Tuner полагался на ручную настройку, а AC/DC предполагал возможность перехвата трафика. Было решено разработать новое средство, позволяющее собирать статистику TCP соединений, оценивать характеристики качества и выбирать оптимальные алгоритмы, основываясь на оценке.

2. Разработанное средство

2.1 Управление перегрузкой в ядре Linux

Для начала рассмотрим детали управления перегрузкой в ядре Linux в нужной для понимания реализации мере.

Каждый алгоритм управления перегрузкой размещается в отдельном модуле ядра. Единственное исключение - алгоритм Reno, который включён в состав самого ядра и будет использован, если другие алгоритмы недоступны. Функциональность любого алгоритма управления перегрузкой представлена структурой `tcp_congestion_ops`, в которой хранятся указатели на операции, осуществляемые данным алгоритмом. Некоторые операции обязательны (например, получение порога медленного старта), другие опциональны (например, реакция на получение подтверждения). Также помимо операций данная структура хранит имя алгоритма, служащее его уникальным идентификатором в ядре. Остальные поля структуры не существенны в данной работе.

Структура `tcp_congestion_ops` доступна из tcp сокета по указателю, а её содержимое является общим для всех сокетов, использующих один и тот же алгоритм управления перегрузкой. Структура операций алгоритма определяется в модуле этого алгоритма, однако для использования в сокете она должна быть доступна и в самом ядре. Для этого ядро Linux экспортирует функции регистрации и отмены регистрации алгоритма управления перегрузкой, которые вызываются модулями соответствующих алгоритмов при подключении к ядру либо отключении от ядра. В этих функциях происходит взаимодействие со списком ядра, где хранятся указатели на все доступные алгоритмы управления перегрузкой.

Кроме самих операций алгоритмам необходимо хранить некоторые внутренние переменные, которые варьируются от алгоритма к алгоритму. Для этого в сокете выделено приватное пространство заданного размера для хранения данных алгоритма. Кроме того, операциям алгоритма управления перегрузкой передаётся указатель на сокет, и таким образом они могут взаимодействовать с упомянутым пространством. Алгоритм используется в сокете следующим образом: сначала вызывается инициализация (где алгоритм может установить значения своих внутренних параметров), затем алгоритм непосредственно занимается управлением перегрузкой, после чего производится отключение алгоритма от сокета, когда он может очистить свои данные. Инициализация и отключение являются опциональными операциями алгоритма.

В пространстве пользователя присутствуют следующие возможности:

- можно выбрать алгоритм управления перегрузкой по умолчанию, который будет использован для всех последующих сессий;
- можно в рамках приложения настраивать алгоритмы управления перегрузкой для сокетов, которые были созданы в данном приложении.

Таким образом, возможности пользователя системы в отношении управления перегрузкой ограничены: например, нет возможности задать индивидуальный алгоритм для выбранного приложения или для соединении с определённым узлом сети.

2.2 Архитектура разработанного средства

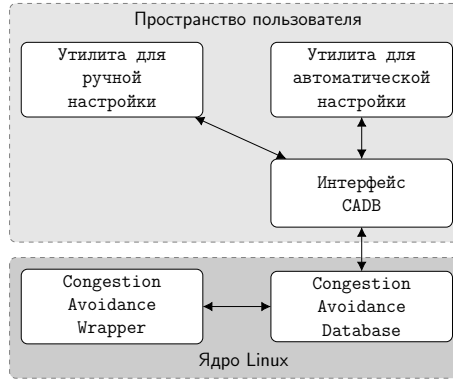

Рис. 1: Компоненты разработанного средства

Разработанное средство включает в себя следующие компоненты (рис.1):

- Congestion Avoidance Database (CADB) – модуль ядра Linux, хранящий информацию об оптимальных алгоритмах управления перегрузкой для потоков трафика определённого вида (с указанными адресами отправителя и получателя);
- Congestion Avoidance Wrapper (CAW) – модуль ядра Linux, позволяющий установить требуемый алгоритм управления перегрузкой на сокет при его инициализации;

- Утилиты пространства пользователя, ответственные за определение оптимального алгоритма управления перегрузкой.

Далее будут подробно рассмотрены компоненты разработанного средства.

2.3 Congestion Avoidance Database

CADB представляет из себя подключаемый модуль ядра Linux. Он хранит набор правил, предназначенных для выбора некоторого алгоритма управления перегрузкой.

Каждое правило включает в себя:

- Уникальный идентификатор правила. Он представлен IP адресами (и масками) отправителя и получателя. Таким образом, можно настроить выбор алгоритма для произвольной пары подсетей. При необходимости, идентификатор может быть расширен, например, включением портов отправителя и получателя или полем приоритета правил.
- Название алгоритма управления перегрузкой. Значение этого поля должно совпадать с тем названием алгоритма, что указано в структуре `tcp_congestion_ops`. Если в ядре нет алгоритма с заданным в правиле названием (ошибка в названии, или модуль с алгоритмом был отключен от ядра), это правило не будет влиять на выбор алгоритма управления перегрузкой.
- Статистика TCP сессий, удовлетворяющих этому правилу. В каждом правиле выделено место для собираемой статистики, включающей в себя число подтверждённых пакетов, долю повторных передач пакетов от общего числа переданных пакетов и оценку RTT.

Интерфейс модуля позволяет добавлять, удалять и редактировать правила. Команды являются экспортируемыми функциями, которые могут вызывать другие модули (например, CA Wrapper). Также был разработан интерфейс для взаимодействия с CADB из пространства пользователя.

2.4 Congestion Avoidance Wrapper

CA Wrapper представляет из себя алгоритм управления перегрузкой, реализованный в подключаемом модуле ядра. Используемое в ядре название данного алгоритма – `tcp_ca_wrapper`.

Разработанный алгоритм обладает рядом отличительных особенностей.

Хранение списка обёрток алгоритмов

Обёртка представляет из себя алгоритм управления перегрузкой, чья структура `tcp_congestion_ops` содержит дополнительное поле с указателем на структуру операций некоторого другого алгоритма. Алгоритм, на который указывает обёртка, далее будет называться встроенным. В обёртке определены все те операции, что и во встроенном алгоритме. Кроме того, в обёртке всегда определён метод `in_ack_event`, используемый для подсчёта числа подтверждённых пакетов. Операции обёртки способны выполнять произвольные действия с сокетом, после чего они вызывают соответствующие операции встроенного алгоритма (если те определены, что важно в случае опциональных операций, таких как `in_ack_event`). CA Wrapper хранит внутри себя список обёрток стандартных алгоритмов управления перегрузкой.

Добавление/удаление обёрток алгоритмов

Чтобы использовать операции алгоритмов управления перегрузкой внутри соответствующих обёрток, необходимо для начала получить доступ к структурам этих алгоритмов. В ядре Linux хранится список указателей на операции алгоритмов, однако он не является экспортируемым объектом и недоступен для подключаемых модулей. Зато доступными являются функции регистрации и отмены регистрации алгоритмов управления перегрузкой, так как эти функции вызываются самими модулями алгоритмов при подключении к ядру или отключении от ядра. Механизм jprobes ядра Linux позволяет вызывать вместе с некоторой экспортируемой функцией ядра другие функции с теми же аргументами и возвращаемым значением. Использование jprobes позволило добавлять в список обёртки алгоритмов, которые проходят регистрацию в ядре, а также удалять обёртки тех, чьи модули отключаются от ядра.

Жизненный цикл tcp_ca_wrapper

Под жизненным циклом подразумевается использование алгоритма в рамках одной TCP сессии. При инициализации TCP сессии с алгоритмом `tcp_ca_wrapper` происходят следующие события:

1. `tcp_ca_wrapper` запрашивает информацию об оптимальном алгоритме управления перегрузкой у модуля CADB. Если получено название алгоритма и найдена его обёртка, она будет использована на следующей стадии. В противном случае будет использована обёртка алгоритма Reno.

2. Алгоритм `tcp_ca_wrapper` подменяет указатель на алгоритм управления перегрузкой в структуре сокета на некоторую обёртку, полученную на предыдущей стадии. Это возможно благодаря передаче указателя на структуру сокета внутрь фукнции инициализации алгоритма управления перегрузкой.
3. Вызывается функция инициализации встроенного алгоритма управления перегрузкой.
4. В предназначенной для хранения состояния алгоритма управления перегрузкой области сокета выделяется место для хранения собираемой во время сессии статистики (включающей в себя те же характеристики, что хранятся в правилах CADB).

За счёт подмены указателя на второй стадии инициализации вместо изначальной структуры `tcp_ca_wrapper` используется выбранная обёртка. Когда вызывается операция обёртки, сначала производится сбор статистики (если он реализован в теле данной функции обёртки), а затем вызывается соответствующая функция встроенного алгоритма, если она определена. Таким образом, управление перегрузкой на самом деле осуществляется встроенным алгоритмом.

В конце сессии статистика передаётся в CADB.

2.5 Утилиты пространства пользователя

В пространстве пользователя производится взаимодействие с правилами CADB при помощи упомянутого ранее интерфейса в виде динамической библиотеки.

Реализация включает в себя две утилиты: утилиту для ручной настройки управления перегрузкой и утилиту для выбора оптимального алгоритма по текущей статистике.

Вторая утилита содержит в себе набор деревьев принятия решений, при помощи которых производится выбор оптимального алгоритма. Каждое дерево связано с одним из стандартных алгоритмов управления перегрузкой. Получив на вход текущую статистику правила с некоторым алгоритмом, средство передаёт статистику в дерево, предназначенное для данного алгоритма. На выходе утилита получает предсказание характеристик качества данного соединения. На основании этого предсказания производится выбор оптимального алгоритма для данных характеристик качества. Для осуществления такого выбора и для обучения деревьев необходимы данные, сбор которых будет описан далее.

3. Сбор данных

Сбор данных преследовал две цели:

1. Определение оптимального алгоритма управления перегрузкой для заданных параметров качества соединения. Для достижения данной цели достаточно было собрать информацию о скорости соединения, так как она служит в качестве критерия оптимальности алгоритма.
2. Определение установленных параметров качества по агрегируемым внутри сокета данным: оценке круговой задержки, числу подтверждённых TCP пакетов и числу посланных повторно пакетов. Эти данные были необходимы для оценки характеристик качества сетевого соединения по собранным в сокете данным.

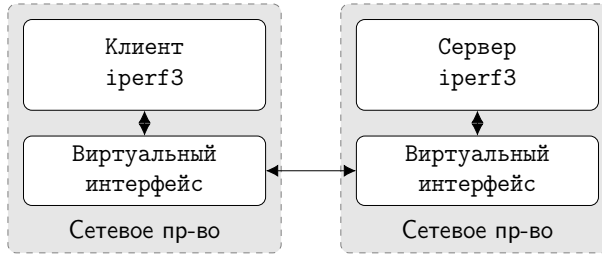

Рис. 2: Топология для сбора данных

Для сбора данных была построена тестовая среда, моделирующая простую топологию из виртуальных сетевых элементов, архитектура которой изображена на рисунке 2.

Сбор данных для определённых условий качества соединения будем далее называть сценарием.

В качестве аргументов сценария выступают параметры качества канала связи между хостами (пропускная способность, процент потерь, задержка, джиттер), используемый алгоритм управления перегрузкой и время передачи данных.

Исходя из поставленных целей, в каждом сценарии собираются следующие данные: средняя скорость TCP соединения, число подтверждённым получателем пакетов, число пакетов, отправленных повторно отправителем и оценка RTT в сокете соединения. Все эти величины рассматриваются на всём временном отрезке сценария.

Конкретные значения указанных ранее аргументов, которые были использованы при сборе данных изображены в таблице 1.

Пропускная способность	$1, 10, 100, 1000$ Мб/с
Потери	$10^{-5}, 10^{-4}, \ldots, 10^{-1}, 1\%$
Задержка	$1, 5, 10, 25, 100$ мс
Джиттер	$0, 10, 20\%$
Алгоритмы	Westwood, Cubic, Reno, Vegas, Illinois, BBR
Время передачи	10с
Время передачи	10с

Таблица 1: Аргументы сценариев

3.1 Собранные данные

Зависимость собранных значений скорости соединения от различных характеристик качества по отдельности показана на графиках рисунка 3. На каждом графике ось ординат отражает среднее значение доли скорости соединения от пропускной способности, а ось абсцисс - значения одной из рассматриваемых характеристик качества соединения.

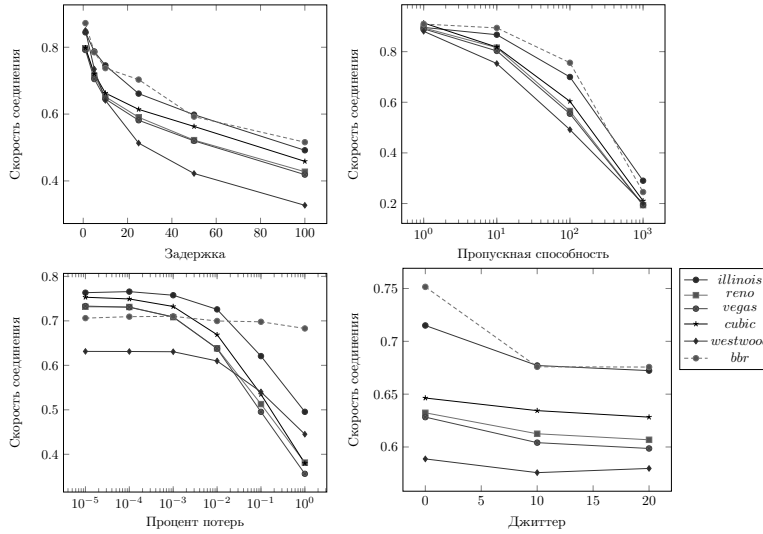

Рис. 3: Зависимость скорости от характеристик качества

4. Экспериментальное исследование

Цель экспериментального исследования – показать увеличение скорости соединения при использовании разработанного средства.

Экспериментальное исследование состоит из ряда сценариев, каждый из которых состоит из двух стадий. Каждая стадия схожа со сценарием при сборе данных. Для проведения экспериментального исследования используется та же топология и те же значения характеристик качества, что и при сборе данных. Цель первой стадии – сбор внутренней статистики сокета. Управление перегрузкой осуществляется предопределённым алгоритмом. После завершения первой стадии собранная статистика используется в описанной ранее утилите пространства пользователя для выбора оптимального алгоритма управления перегрузкой. Выбранный алгоритм используется для управления перегрузкой на второй стадии. Результаты обеих стадий сохраняются, после чего производится сравнение средних показателей скорости соединения начального алгоритма и выбранного.

Важную роль в выборе оптимального алгоритма играет правильная оценка характеристик качества по собранной статистике сокета. Результаты оценки показаны на рисунке 4.

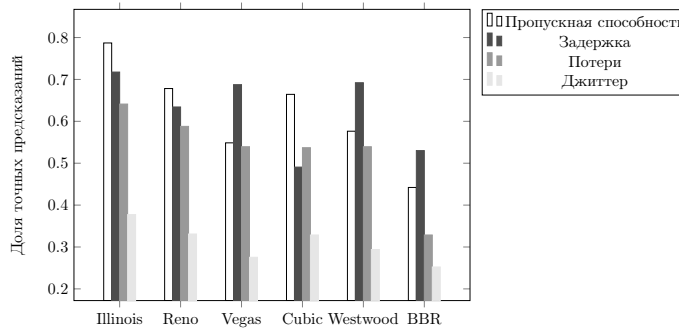

Рис. 4: Точность определения характеристик качества

Данные результаты показывают низкую точность оценки джиттера. Кроме того, все оценки характеристик качества являются наименее точными в случае алгоритма BBR. Например, заметна значительная разница в точности оценки потерь между BBR и другими алгоритмами. Это связано с отсутствием прямой реакции на потери в BBR, что заметно на графиках сбора данных.

Также было произведено сравнение средней скорости при исполь-

зовании изначального алгоритма и выбранного средством. Результаты показаны в таблице 2. Также в таблице показан максимальный возможный прирост для данного алгоритма, который основан на средних значениях скорости соединения, полученных при сборе данных.

	Реальный прирост(%)	Макс. прирост(%)
Illinois	3.15	4.91
Vegas	10.27	12.69
Reno	10.11	12.00
Westwood	12.89	15.58
BBR	-0.85	3.62
Cubic	7.92	10.08

Таблица 2: Изменение скорости соединения

На большинстве алгоритмов был получен значительный прирост за исключением алгоритма BBR, что связано с низкой точностью оценки характеристик качества для этого алгоритма.

Как видно из таблицы 2, на каждом из алгоритмов существует теоретическая возможность получить прирост скорости соединения. И несмотря на то, что предложенный в работе подход к измерению характеристик сетевого окружения довольно примитивен, он позволяет добиться ощутимого прироста на большинстве протестированных конфигураций. Использование более репрезентативного набора данных и более точных методов оценки сетевого окружения потенциально позволит приблизиться к значению максимального прироста.

5. Заключение

В работе был предложен подход к оценке характеристик качества соединения для выбора оптимального алгоритма управления перегрузкой. Экспериментальное иследование показало превосходство разработанного средства перед использованием предопределённых алгоритмов для разных характеристик качества.

Дальнейшие темы исследования включают в себя анализ влияния справедливости TCP потоков, использование более продвинутых подходов к оценке характеристик качества, изменение алгоритмов в течение TCP сессии и проведение экспериментального исследования в реальных сетях.

Литература

1. Miller K., Hsiao L. W. *TCPTuner: Congestion Control Your Way* //arXiv preprint arXiv:1605.01987. – 2016.

2. He K. et al. *AC/DC TCP: Virtual congestion control enforcement for datacenter networks* //Proceedings of the 2016 ACM SIGCOMM Conference. – ACM, 2016. – Pp. 244-257.

3. Winstein K., Balakrishnan H. *TCP ex machina: computer-generated congestion control* //ACM SIGCOMM Computer Communication Review. – ACM, 2013. – Vol. 43. – №. 4. – Pp. 123-134.

4. Alizadeh M. et al. *Data center tcp (dctcp)* //ACM SIGCOMM computer communication review. – 2011. – Vol. 41. – №. 4. – Pp. 63-74.

5. Floyd S., Henderson T., Gurtov A. *The NewReno modification to TCP's fast recovery algorithm.* – 2004. – №. RFC 3782.

6. Ha S., Rhee I., Xu L. *CUBIC: a new TCP-friendly high-speed TCP variant* //ACM SIGOPS operating systems review. – 2008. – Vol. 42. – №. 5. – Pp. 64-74.

7. Liu S., Başar T., Srikant R. *TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks* //Performance Evaluation. – 2008. – Vol. 65. – №. 6-7. – Pp. 417-440.

8. Brakmo L. S., Peterson L. L. *TCP Vegas: End to end congestion avoidance on a global Internet* //IEEE Journal on selected Areas in communications. – 1995. – Vol. 13. – №. 8. – Pp. 1465-1480.

9. Casetti C. et al. *TCP Westwood: end-to-end congestion control for wired/wireless networks* //Wireless Networks. – 2002. – Vol. 8. – №. 5. – Pp. 467-479.

10. Cardwell N. et al. *BBR: congestion-based congestion control* //Communications of the ACM. – 2017. – Vol. 60. – №. 2. – Pp. 58-66.

11. Fu C. P., Liew S. C. *TCP Veno: TCP enhancement for transmission over wireless access networks* //IEEE Journal on selected areas in communications. – 2003. – Vol. 21. – №. 2. – Pp. 216-228.

12. Tan K. et al. *A compound TCP approach for high-speed and long distance networks* //Proceedings-IEEE INFOCOM. – 2006.

Королев В.В., Шалимов А.В.

Разработка системы отображения произвольного конвейера OpenFlow таблиц в коммутаторы с единственной таблицей

Введение

Программно-конфигурируемая сеть (ПКС) — подход, в котором уровень управления отделён от уровня передачи данных. Есть логически централизованный контроллер, который имеет контроль над другими сетевыми устройствами. Одним из способов реализации технологии ПКС является протокол OpenFlow. Одним из базовых факторов функционирования ПКС является наличие Flow-таблиц, где записаны правила, применимые к обрабатываемым сетевым пакетам. Чем больше таких таблиц на коммутаторах, тем более функциональными и эффективными будут реализованные на них программы по управлению сетью. На данный момент пока не появилось полноценных аппаратных OpenFlow коммутаторов.

Логичным выходом из ситуации становится использование нынешних классических коммутаторов, для которых OpenFlow таблицы оттранслированы в небольшое и удобное число физических таблиц на коммутаторах. Данный подход позволит ускорить обработку пакетов данных, при этом не вмешиваясь в физическое устройство коммутаторов, к тому же благодаря объединению таблиц можно достичь совместимости между сетевым оборудованием разных производителей, которая обеспечивается процессом совмещения правил в единый формат.

1. Цель работы и постановка задачи

Целью данной работы является разработка системы отображения произвольного набора OpenFlow таблиц, состоящих из правил, в одну общую на сетевом устройстве. Система должна поддерживать правила, содержащие три основных поля: поле для сравнения, поле действий, приоритет (см. 4.1) — к тому же, обрабатывать добавления и удаления правил.

Постановка задачи: разработать, реализовать и исследовать алгоритм трансляции множества OpenFlow таблиц в единственную таблицу в сети.

2. Обоснование актуальности

1. Как правило, до сих пор наиболее распространены классические коммутаторы с OpenFlow прошивкой, где имеется небольшое число OpenFlow таблиц – одна или две.

2. Даже при разработке OpenFlow[4] коммутатора с нуля на базе сетевых процессоров возникают проблемы с реализацией нескольких OpenFlow таблиц. Чем больше таких таблиц, тем больше дополнительных проходов через сетевой процессор должен совершить пакет, что в результате уменьшает итоговую пропускную способность устройства.

3. Обзор существующих подходов

3.1 Язык программирования Pyretic

Pyretic[3] позволяет программистам сосредоточиться на том, как задать сетевую политику на высоком уровне абстракции, вместо того, чтобы думать о том, как внедрить ее используя низкоуровневые механизмы OpenFlow. Pyretic вместо внедрения политики путем установки физических правил одного за другим на коммутаторах устанавливает единую политику для всей сети посредством функций от входящего пакета. На выходе могут быть модифицированные поля пакета, но программист уже не беспокоится о том, какие именно OpenFlow правила были применены при прохождении пакета.

Одно из преимуществ политик Pyretic как абстрактных функций – это модульное программирование. Традиционно в ПКС нельзя написать модули приложения независимо, так как могут быть пересечения, Pyretic же собирает различные политики вместе при помощи операторов композиции политик (например, параллельная композиция или последовательная композиция). Для мониторинга состояния сети и сбора статистики Pyretic имеет поддержку высокоуровневых запросов. К тому же, могут создаваться и динамические политики, к которым так же возможно применение операторов композиции.

Сетевая политика как функция

На входе — пакет из заданного местоположения (коммутатор и порт), на выходе – новые пакеты с возможно разными местоположениями. Возвращение пустого набора пакетов – сброс пакета. Если на выходе лишь один пакет – отправка пакета в новое расположение.

Если же на выходе множество пакетов, то это мультикаст. Фрагмент политики, которая использует некоторые возможности Pyretic, чтобы провести пакет, имеющий ip назначения 10.0.0.1 через коммутаторы A и B (Рис.1):

```
(match(switch=A) & match(dstip='10.0.0.1') >> fwd(6)) +
(match(switch=B) & match(dstip='10.0.0.1') >> fwd(7))
```

Рисунок 1. Пример политики Pyretic

Здесь используются предикатные политики (match и конъюнкция), чтобы отсеивать пакеты по их расположению и содержимому, также используются политики модификации (fwd), чтобы обрабатывать пакеты и направлять в нужном направлении, и операторы композиции (+ как параллельная и » как последовательная).

Политики внешне похожи на OpenFlow правила своими компонентами, так как отделяют по параметрам некоторые пакеты и выполняют различные действия. Из-за такого подхода программистам надо переключиться с правил, как основы, на функциональный подход к созданию приложений. Программист думает об высокоуровневой логике организации приложения, а не о низкоуровневом написании этой логики с учетом физического оборудования. Самая простая политика Pyretic записывается следующим образом: flood(). Она означает, что каждый коммутатор отправляет пакет на все порты остовного дерева сети. В традиционном OpenFlow программировании приложение контроллера должно было бы для каждого коммутатора в сети вызвать flow-modification для установки правила со следующим паттерном: don't care на всех битах, с одним единственным действием flood (если такое действие поддерживается коммутатором). Табл.1 с некоторыми общими базовыми политиками Pyretic приведена ниже:

Syntax	**Summary**
identity	returns original packet
none	returns empty set
match(f=v)	identity if field f matches v, none otherwise
modify(f=v)	returns packet with field f set to v
fwd(a)	modify(port=a)
flood()	returns one packet for each local port on the network spanning tree

Таблица 1. Базовые политики Pyretic

Среда выполнения Pyretic включает в себя интерпретатор, сравнивающий входящий пакет с текущей политикой. В простом режиме операций все пакеты изначально оцениваются интерпретатором. Тем временем, среда выполнения отслеживает активные запросы (высокоуровневые конструкции Pyretic, например: $Q = packets(limit = 1, groupby = [\prime srcip\prime])$), обновления динамических политик и изменения топологии сети. Среда выполнения "реактивно" устанавливает правила на коммутаторы, когда это можно сделать безопасно, чтобы обработать будущие пакеты, которые подойдут под оценочные критерии (например, пакеты от одного и того же TCP соединения).

Вывод: Pyretic для рассматриваемой задачи не применим. В нем используются статичные политики без динамического добавления/удаления правил и приложений (предполагается иметь такую возможность). К тому же, Pyretic позволяет обрабатывать пакет при помощи модульного и функционального программирования. Данный факт мешает прозрачности для программиста. Такой подход вынуждал бы многих разработчиков изучать узкоспециализированный функциональный язык.

3.2 Технология TableVisor

Основная идея в том, что несколько физических коммутаторов могут быть объединены между собой, чтобы эмулировать единое устройство с расширенными возможностями и емкостью. Может использоваться, к примеру, для построения многофункциональных многотабличных конвейеров из конфигураций коммутаторов, имеющих разный функционал. Могут использоваться в инфраструктуре даже коммутаторы, поддерживающие только одну таблицу правил.

Реализован как прокси-сервер между контроллером ПКС и сетевыми физическими устройствами ПКС. Сообщения протокола OpenFlow, отправленные либо контроллером, либо устройствами, перехватываются, модифицируются на уровне TableVisor[2], чтобы реализовать абстракцию, в которой множество коммутаторов представляется контроллеру как один.

TableVisor[1] ведет себя как эмулированный коммутатор на взгляд контроллера и как контроллер для используемых физических коммутаторов. Контроллер OpenFlow не знает ничего про многотабличный конвейер. Такое устройство уровней: уровень коммутатора, уровень контроллера и уровень обработки сообщений – упрощает разработку, поддержание соединения, а также отладку ошибок[2].

Уровень коммутатора отвечает за установление соединения с контроллером. Именно этот уровень позволяет вести себя как один OpenFlow коммутатор.

Обработка сообщений затрагивает все сообщения, идущие от контроллера к коммутаторам и обратно. Содержимое сообщений изменяется таким способом, что обыкновенные сообщения OpenFlow от контроллера могут доходить в необходимом виде до коммутаторов. Процесс изменения сообщений — это основополагающая функциональность TableVisor[1].

Уровнем контроллера является уровнем TableVisor, к которому подключены коммутаторы. IP-адрес реального контроллера по этой причине должен быть заменен адресом хоста, на котором запущен TableVisor. Тогда коммутаторы видят "уровень контроллера"как обыкновенный OpenFlow контроллер. Коммутаторы в данной ситуации не знают о многотабличной архитектуре, используемой TableVisor, так как каждый из них имеет только одну таблицу. Контроллер указывает TableVisor использовать несколько таблиц при помощи goto-table инструкции, которая обрабатывается на уровне обработки сообщений и отправляется к соответствующему коммутатору как выходное действие.

Для реализации прозрачного прокси-сервера нужно преобразовывать сообщения, проходящие через него в обоих направлениях. Этим занимается уровень обработки сообщений TableVisor. Чтобы этого добиться, сообщения между контроллером и физическими коммутаторами перехватываются, преобразуются и отправляются к коммутаторам и контроллеру соответственно[5].

TableVisor представляет собой прокси-сервер, написанный на распространенном и известном многим разработчикам языке Java[5], имеет возможность эмулировать коммутатор с большим функционалом за относительно малую финансовую сумму по сравнению с физическим коммутатором, имеющим необходимый функционал. Также обеспечивает работу в одной системе устройств разных производителей. Нашу задачу напрямую не решает, так как нет единой обобщенной таблицы в каждый момент времени, но использует полезный подход прокси-сервера.

3.3 Выводы

Из рассмотренных существующих решений в данной области можно сделать вывод, что нет четких подходов к решению поставленной задачи, но для решения можно использовать подходы Pyretic, в частности, объединение его политик в одну таблицу, и подход TableVisor в виде прокси-сервера.

4. Предлагаемое решение

4.1 Структура правил OpenFlow

Для начала рассмотрим классический вид правила OpenFlow (Табл.2):

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Таблица 2. Структура правила OpenFlow

Основные составляющие правила: поля для сравнения, приоритет, счетчики, инструкции, таймауты, куки, флаги[4].

- Поля для сравнения: по ним выбирается подходящий под это правило пакет. Составляющими являются входной порт и заголовки пакета и возможно другие поля конвейера такие, как метаданные, установленные предыдущей таблицей.
- Приоритет: соответствие приоритетности входного потока.
- Счетчики: обновляются, когда пакет подходит под правило.
- Инструкции: изменение набора действий.
- Таймауты: максимальное количество времени, в течение которого правило может не использоваться. По истечении будет удалено коммутатором.
- Куки: скрытые значения некоторых данных, подобранные контроллером. Могут быть использованы контроллером для фильтрации некоторых правил после запросов модификации/удаления правил. Не используются во время обработки пакетов.
- Флаги: изменяют способ обработки потока пакетов, присутствуют предустановленные переменные, изменение значения которых, модифицирует правила, применяемые к данному пакету.

4.2 Структура правил алгоритма

В разрабатываемом алгоритме важен принцип и понимание правильного формирования единой таблицы правил, в связи с чем было решено рассматривать предельно понятный и информативный вид правил, так как добавление каких-либо элементов к нему не является

задачей алгоритмически трудной, но усложняет понимание процесса. Правила, рассматриваемые в алгоритме, имеют лишь: поля для сравнения, инструкции и приоритет (Табл.3).

Приоритетом является некоторое целое число. Поля для сравнения и инструкции (далее именуемые как действия) имеют схожую структуру: это множества, состоящие из пар, где первый элемент пары – название поля, а второй элемент – значение рассматриваемого поля.

Match Fields	Instructions	Priority

Таблица 3. Выбранные поля

Поля для сравнения несут в себе информацию: подходит ли пакет к данному правилу; действия, в свою очередь, показывают, как преобразовать указанное поле.

Действия в общем случае могут быть apply-action и write-action. Первое, согласно своему названию, модифицирует пакет сразу, как только правило применяется к пакету. Таким образом, при прохождении через ряд таблиц к одному и тому же пакету может быть применено множество действий на каждом прохождении. Второй же тип действий заключается в том, что не происходит моментального применения действия из соответсвующего правила, а действия записываются во множество действий и применяются только тогда, когда запись более не возможна, так как пакет прошел через все необходимые таблицы. При добавлении во множество действий, если такое уже присутствует во множестве, но с другим значением, то значение заменяется на добавляемое. По этой причине множество можно считать уникальным по названию полей, над которыми проводятся действия.

Для разработки алгоритма было решено рассматривать лишь действия типа write-action, которые являются обязательными по спецификации OpenFlow. Действия типа apply-action, являющиеся опциональными в спецификации OpenFlow, в данной работе не рассматриваются в силу усложнения алгоритма работы.

Стоит выделить пару особых правил, которые допустимы в любой таблице. Такие правила имеют самый маленький приоритет, в поле для сравнения отсутствуют значения (можно обозначать *), что значит, что это правило применимо к любому пакету. В действиях может располагаться либо drop (сбросить пакет), либо goto-next-table, что означает перейти к сравнению полей пакета с правилами из следующей таблицы. Из этого вытекает, что последняя таблица не может иметь особого правила с действием goto-next-table, так как

после нее не идет никакой таблицы. Эти правила имеют самый низкий приоритет по следующей причине: если их приоритет не минимален, то есть правила с меньшим приоритетом, но при просмотре подходящих правил для пакета сверху вниз в таблице по убыванию приоритетов возможно всегда применение особого правила, так как допускает любое поле с любым значением, из чего вытекает неприменимость правил, расположенных ниже. Делаем вывод, что не целесообразно иметь в таблице правила с приоритетом меньшим, чем у особых. Так как эти правила расположены всегда в конце таблицы и имеют определенный вид (Табл.4):

*	drop	0
*	goto-next-table	0

Таблица 4. Правила drop и goto-next

Можем не рассматривать их в виде правил, а уточнять лишь тип таблицы. Таблицы с drop обозначим через D, таблицы с goto-next-table обозначим через G.

4.3 Описание алгоритма

Входные параметры предлагаемого алгоритма: набор заданных таблиц, проходим по нему справа налево. Обход таблиц от первой до последней усложняет алгоритм из-за наличия таблиц типа G. Каждый раз рассматриваются лишь две таблицы: сформированная на прошлом шаге с текущей (на первом шаге сформированная совпадает с самой последней таблицей), на последнем шаге будет сформирована необходимая единая таблица. Тип формируемой таблицы на каждом шаге совпадает с типом текущей таблицы (левая из рассматриваемых). Если тип текущей таблицы D, то формируемая таблица — это объединение рассматриваемых; если тип G, то формируемая — объединение рассматриваемых плюс приписывание вниз уже сформированной на предыдущем шаге. При объединении двух таблиц смотрим на пары правил: фиксируем верхнее из левой таблицы сравниваем со всеми из правой поочередно сверху вниз, потом фиксируем второе из левой таблицы — со всеми из правой, и так далее. В итоге, реализуется декартово произведение правил таблиц: порядок обхода задан, сравним каждое правило с каждым. При рассмотрении полей для сравнения (match fields) двух правил находим пересечений названий: если пусто или есть пересечение, но с равными значениями, то добавляется правило с объединением полей для сравнения, если же в пересечении различные значения соответствующих полей, то новое правило в единую таблицу не добавляется.

При добавлении нового правила поле действий (action) будет формироваться так: добавляется объединение действий рассматриваемых правил с учетом того, что при непустом пересечении в объединение войдут поля со значением из второго правила. В итоговой таблице приоритеты расставляются снизу-вверх по возрастанию, что можно сделать благодаря предложенному способу обхода таблиц и порядку добавления правил. «Специальным объединением» двух таблиц будем называть такую таблицу, которая построена вышеописанным способом.

Для учета типов таблиц был разработан следующий пункт алгоритма:

- Пусть рассматриваются две таблицы T_k и T_{k+1}
- Если тип таблицы T_k == D (drop), то формируемая таблица CT (combined table) будет представлять собой «специальное объединение» таблиц T_k и T_{k+1} и будет иметь тип D (drop).
- Если же тип таблицы T_k == G (goto-next-table), то формируемая таблица CT (combined table) будет не только представлять собой «специальное объединение» таблиц T_k и T_{k+1}, но и будет иметь приписанные под получившимися правилами все правила таблицы T_{k+1}. Тип получившейся CT (combined table) будет G.

Предложенный выше способ обхода таблиц был разработан из следующих рассуждений: пусть все правила итоговой таблицы сформированы из каких-то правил, соответствующих некоторым таблицам. Тогда в формировании каждого правила участвовали все таблицы типа D, а оставшиеся таблицы типа G могли входить или не входить в формирование. Из-за чего так происходит? Итоговая таблица должна отображать правила, аналогичное применение которых в варианте со множеством таблиц провело бы пакет от первой до последней таблицы. В формировании всегда есть таблица типа D, потому что её отсутствие означает, что никакое правило из этой таблицы не было бы применено в начальном варианте, тогда бы пакет сбросился. Таблицы типа G могут входить или нет в формирование, так как вхождение означает использование правила таблицы в начальном варианте, а отсутствие соответствует прохождению в начальном варианте этой таблицы по правилу goto-next-table. Таким образом, каждому правилу итоговой таблицы можно сопоставить бинарный набор, в котором номера позиций соответствуют номерам таблиц в исходной конфигурации, наличие 1 на позиции говорит о том, что использовалась эта таблица в формировании правила, 0 — что не

использовалась. На позициях таблиц типа D всегда расположены 1, на оставшихся позициях нужно перебрать все возможные комбинации, упорядочим лексикографически. Предложенный обход реализует итоговую таблицу, которую можно получить описанным методом с бинарными наборами. Стоит уточнить, что 1 в наборах говорят лишь о том, что нужно рассмотреть правила, после рассмотрения и применения описанного алгоритма новые правила могут быть не сформированы.

5. Формальное описание алгоритма

5.1 Обозначения

Дано упорядоченное конечное множество $T = \{T_1, T_2, T_3, \ldots, T_n\}, |T| = n$

Элемент множества назовем таблицей (Table).

- T_i, где $i = (1, ..., n)$, является упорядоченным конечным множеством из кортежей. (T_1 является представлением крайней правой таблицы)
- R_{ij}, где $j = (1, ..., |T_i|)$. Кортеж назовем правилом (Rule).
- R_{ij} является упорядоченной тройкой M, A, P , где M – поле для сравнения (Match field), A – действие (Action), P – приоритет (Priority).
- P есть натуральное число;
- M – множество пар вида <F, V>, где F – название поля для сравнения (Field), V – значение соответствующего поля (Value);
- A – множество пар вида <F, H>, где F – название поля, над которым должно быть совершено действие (Field), H – число, которым закодировано действие.

M и A полностью идентичны по структуре, но имеют разный смысл: M - поля сравнения, A - поля действий.

Прим.: drop и goto-next-table определяют тип кортежа, что влияет на общий порядок обхода и формирования таблиц, описанный в предыдущих разделах.

5.2 Основной шаг сравнения

- S-1 таблицы уже рассмотрены
- Рассматриваются CT, T_s и формируется NCT (New Combined Table).
- Сортируются R_{sj} в порядке убывания приоритета.
- Далее сравниваются R_{CTm} и R_{sk}, где $m = (1, ..., |CT|), k = (1, ..., |T_s|)$.

  Итерация по m осуществляется только после перебора k. Процесс сравнения R_{CTm} и R_{sk} состоит в следующем:

$M_{CTm} \cap M_{sk}$ по F, т.е. выбираем пересечение пар только по названию поля. Если пересечение пусто, то добавляется кортеж R в NCT, у которого $M = M_{CTm} \cup M_{sk}$. Если пересечение не пусто, то, если существуют V для пересекающихся F не равные, нет добавления в NCT, потому что мы пытаемся совместить два правила, которые ищут совпадение по одному и тому же полю, но с разными значениями. Если же все V для пересекающихся F имеют одинаковое значение, то добавляется R в NCT, у которого $M = M_{CTm} \cup M_{sk}$.

- CT удаляется. NCT обозначается через CT.

5.3 Приоритеты и действия

При рассмотрении двух кортежей i и j делаем следующее:
$A_i \cap A_j$ по F, т.е. выбираем пересечение пар только по названию поля. Если пересечение пусто, то получается $A = A_i \cup A_j$. Если пересечение не пусто, то, если существуют H для пересекающихся F не равные, то $A = A_i \cup A_j$ без пар <F, H>, которые были в пересечении со значениями из A_i, если же все H для пересекающихся F имеют одинаковое значение, то $A = A_i \cup A_j$.

- P последнего кортежа CT получает приоритет = 1, предпоследнего = 2, следующего = 3, ... , первого = $|CT|$.

6. Экспериментальное исследование

Для проведения экспериментов и анализа разработанного алгоритма была написана программа на языке C++. Входными параметрами служат различные текстовые файлы формата .txt, содержащие внутри себя таблицы разных размерностей, задающие конфигурацию. Под конфигурацией понимается набор полей в полях

для сравнения и полях действий: могут пересекаться и иметь разные значения, могут пересекаться и иметь одно значение, могут не пересекаться. К тому же, был рассмотрен различный порядок типов таблиц, например, DDGGDD, GGDDDGD, GDGDGD, DGDD, GGGGDD, DDDD, GDGGDDD. Корректность работы разработанного алгоритма подтверждается теоретически, так как строятся правила для всех возможных случаев прохождения пакета по данной изначально конфигурации и соблюдается приоритет, о чем говорилось выше. Кроме того, корректность подтверждалась на рассмотренных 20 случаях после ручной проверки. Во всех рассмотренных случаях итоговая таблица была построена верно. Можно заметить, что количество правил итоговой таблицы в разы отличается от того количества, которое имеет место в случае множества таблиц (Табл.5).

На основании исследования можно сделать вывод, что при увеличении общего количества правил из данного множества таблиц, больше становится экспоненциальный взрыв, иными словами, чем больше значений приходится перебирать алгоритму, тем быстрее увеличивается количество правил в итоговой таблице. Есть пути оптимизации. Например, удаление повторяющихся правил. Стоит заметить, что количество правил в итоговой таблице зависит не только от суммарного количества правил в начале, но и ещё от самих этих правил, чем больше пересечений по названию полей, тем больше вероятность того, что значения этих полей различны, и правило не будет добавлено в итоговую таблицу.

Количество таблиц	Суммарное количество правил таблиц	Количество правил в итоговой таблице
3	20(6, 6, 8)	307
5	20(4, 4, 4, 4, 4)	373
4	20(5, 5, 6, 4)	652
3	40(17, 13, 10)	2038
4	50(4, 16, 15, 15)	7481
4	50(15, 15, 16, 4)	7293
4	60(20, 3, 4, 33)	7700

Таблица 5. Результаты экспериментов

7. Заключение

Программно-конфигурируемые сети решают некоторые проблемы традиционных сетей, из-за чего сфера применения ПКС только разрастается, сюда можно включить центры обработки данных, домашние сети, телекоммуникационные сети. Сфера становится шире, а это значит, что нужно писать больше приложений, для чего хочется иметь достаточно вариативные условия написания.

Разработанный алгоритм отображения множества OpenFlow таблиц в одну может расширить функциональность TableVisor, если будет внедрен между OpenFlow контроллером и TableVisor прокси-сервером. Алгоритм был разработан с принятием некоторых ограничений: рассмотрение только write-action действий, отсутсвие предположений о хранении данных.

Следующими шагами по улучшению алгоритма планируется реализация поддержки apply-action действий, проведение тестирования на реальных данных, интеграция в TableVisor, удаление повторяющихся правил в итоговой таблице, оптимизация обработки правил.

Литература

1. Gebert S. *Tablevisor: An emulation layer for multi-table openflow switches.* Software Defined Networks (EWSDN), 2015 Fourth European Workshop on. – IEEE, 2015. – С. 117-118.

2. Geissler S. *Tablevisor 2.0: Towards full-featured, scalable and hardware-independent multi-table processing.* Network Softwarization (NetSoft), 2017 IEEE Conference on. – IEEE, 2017. – С. 1-8.

3. Reich J. *Modular sdn programming with Pyretic.* Technical Report of USENIX. – 2013.

4. Nygren A. *OpenFlow Switch Specification, Version 1.3. 4 (Protocol version 0x04), Mar. 27, 2014* Open Networking Foundation.(Part 1 of 2). – С. 1-84.

5. The continued TableVisor project – JTableVisor: `https://github.com/lsinfo3/JTableVisor`

Костенко В.А., Маслов Н.С.

Исследование влияния способа выбора начального приближения на точность и вычислительную сложность обучения нейронных сетей

Введение

Методы машинного обучения позволяют решать задачи, для которых сложно предложить аналитическую модель или её применение затруднено из-за высокой вычислительной сложности. Одним из наиболее часто используемых классов моделей являются искуственные нейронные сети.

Качество работы нейронной сети зависит от архитектуры и значений весов связей между нейронами. Для обучения нейронных сетей широко используются локально-оптимальные алгоритмы [1], для которых требуется начальное приближение. Для некоторых классов нейронных сетей были предложены алгоритмы, позволяющие найти "хорошее"начальное приближение на основе тренировочной выборки [2], но в большинстве случаев на практике оно часто выбирается случайным образом [3-5].

Данная статья посвящена исследованию зависимости качества работы обученной нейронной сети от выбора способа инициализации весов на примере задач классификации изображений [6-8]. Обучение сетей производится методом стохастического градиентного спуска с фиксированным шагом обучения [9], а также с помощью адаптивного алгоритма AdaDelta [10].

1. Особенности искуственных нейронных сетей

Нейронная сеть прямого распространения - это модель, основанная на представлении об устройстве и функционировании биологических нейронных сетей.

В качестве модели нейрона используется сумматор со взвешенными связями со всеми элементами входного вектора слоя. Для того, чтобы придать нейронной сети свойство нелинейности, значение суммы проходит через нелинейную активационную функцию. Набор выходов всех нейронов формирует выходной вектор.

Нейроны в сети группируются в слои. Отдельно выделяется входной слой, нейроны которого получают непосредственно набор входных значений, выходной слой и скрытые слои.

Полносвязный слой - это набор нейронов, каждый из которых соединён взвешенной связью с каждым входным сигналом. Нейронная сеть, состоящая из последовательности полносвязных слоёв, соединённых друг с другом в цепь, называется полносвязной сетью прямого распространения. Этот вид сетей является одним из самых старых и хорошо изученных; доказано, что полносвязная сеть с одним скрытым слоем является универсальным аппроксиматором [11].

Полносвязные сети неплохо подходят для данных фиксированного размера в общем, но для некоторых особых видов данных (например, для изображений и временных рядов) существуют другие нейронные структуры, более предпочтительные по свойствам и вычислительной сложности.

Для обработки изображений и временных рядов используют свёрточные нейронные сети [1].

Одна из неформальных интерпретаций операции свёртки - схожесть одной функции с отражённой и сдвинутой копией другой. В задаче распознавания изображения это свойство используется для поиска признаков на изображении. Например, если определить функцию $f(x, y, z)$ как исходное изображение, а $g(x, y, z)$ как искомую форму (например, изображение вертикальной линии), то на изображении $C(x, y) = (f * g)(x, y)$ наибольшие значения будут в тех областях, где расположены вертикальные линии.

Особенность свёрточных нейронных сетей заключается в повторном использовании весов для разных элементов входных данных. Например, при обработке изображения свёрточный слой может искать определённый признак - прямую линию, переход цвета и т.п., при этом поиск производится сразу по всему изображению сканированием. Для этого требуется всего одна небольшая матрица свёртки значительно меньшего размера, чем само изображение. Помимо уменьшения количества весов, эта особенность позволяет улучшить обобщающую способность, устраняя зависимость между признаком и его расположением на изображении.

2. Обучение нейронной сети

Обучение нейронной сети - поиск таких значений весов, при которых минимизируется значение функции ошибок на тренировочной выборке. Наиболее часто для обучения используются алгоритмы, основанные на методе обратного распространения ошибки, идея кото-

рого заключается в пошаговой коррекции весов в направлении градиента функции ошибки (что накладывает требование дифференцируемости). Подробное описание метода обратного распространения ошибки приведено в книге [12].

Метод обратного распространения является разновидностью градиентного спуска и обладает характерными для него проблемами:

- локальность - алгоритм нацелен на поиск локального минимума, результат работы существенно зависит от начальной точки;
- проблема выбора шага обучения: при слишком маленьком шаге обучение может занимать неприемлемо много времени, при слишком большом возможны пропуски ("перескакивания") минимумов.

Одной из проблем метода обратного распространения на основе градиентного спуска является относительно низкая скорость обучения, поскольку для изменения весов требуется вычислить градиент на всей обучающей выборке. Для ускорения обучения было предложено использовать стохастический градиентный спуск (SGD) [9].

Идея стохастического градиентного спуска заключается в обновлении весовых коэффициентов после вычисления градиентов для набора $n < m$ случайных элементов из обучающей выборки. Коррекция весов считается точно так же, как в случае с градиентным спуском.

AdaDelta [10] - алгоритм оптимизации, представляющий собой SDG с автоматически выбираемым шагом обучения.

AdaDelta основан на алгоритме AdaGrad, суть которого заключается в выборе шага обучения для каждого отдельного параметра, обратно пропорциональным сумме квадратов его обновлений за всё время обучения. Это позволяет "плавно"обучать часто обновляемые параметры, и при этом сохранять чувствительность для тех параметров, которые обновляются редко. Отличие AdaDelta заключается в том, что вместо суммы квадратов обновлений используется затухающее среднее, что позволяет избежать паралича сети для очень часто обновляемых параметров.

3. Методы инициализации весовых коэффициентов

Обучение нейронной сети методом обратного распространения ошибки требует начальной инициализации весов сети. От выбора

начальной точки зависит сходимость метода в целом, достижимое минимальное значение ошибки и реальная обобщающая способность сети при одинаковых значениях ошибки на обучающей выборке.

Для решения этой проблемы предпринимались попытки создания специальных алгоритмов. Например, в работе Jim Yan, Tommy Chow [2] был предложен метод инициализации весов для полносвязных нейронных сетей на основе обучающей выборки. Однако, данный метод накладывает набор ограничений на архитектуру сети и неприменим для других видов нейронных сетей (например, свёрточных). Более того, некоторые точки, которые хороши с точки зрения оптимизации, не придают нейронной сети обобщающих свойств (переобучение). В силу этих причин метод не получил большого распространения на практике.

На сегодняшний день не было предложено качественной теории, описывающей, каким образом инициализировать веса нейронной сети. Основное известное правило инициализации весов заключается в устранении симметрии между нейронами. Это означает, что при инициализации весов в одном слое не должно оказаться нейронов с одинаковыми весами, поскольку при обучении детерминированным алгоритмом с детерминированной функцией потерь их веса будут изменяться одинаковым образом.

Правило инициализации весов в нейронной сети поощряет использование максимально различных функций. Одним из способов гарантированно получить различные функции на разных нейронах заключается в использовании ортогональной матрицы весов [5].

Если количество строк матрицы весов не больше количества столбцов (то есть, количество нейронов слоя не больше количества входов), то возможно построить матрицу с ортогональными строками. Для построения такой матрицы можно использовать процесс ортогонализации Грама-Шмидта на основе случайно сгенерированной матрицы с линейно независимыми строками.

Преимущество данного подхода заключается в гарантированном получении существенно различающихся функций нейронов на этапе инициализации. Недостаток заключается в вычислительной сложности генерации орготональной матрицы: например, процесс Грама-Шмидта имеет последовательную сложность $O(nm^2)$, где n - число входных сигналов слоя, m - количество нейронов на слое.

Правило устранения симметрии мотивирует использовать случайные значения для инициализации весов. Преимущество этого подхода по сравнению с ортогонализацией заключается в меньшей вычислительной сложности. При использовании источника случайных значений с достаточно большой энтропией в многомерном про-

странстве вероятность получить линейно зависимые векторы крайне низкая. Это позволяет с достаточно большой уверенностью использовать метод инициализации случайными значениями для нейронных сетей.

Параметры распределения случайных значений, используемых для инициализации весов нейронной сети, могут существенно повлиять на процесс обучения.

С одной стороны, при увеличении абсолютных значений весовых коэффициентов усиливается эффект устранения симметрии, что позволяет избежать появления избыточных нейронов. Также большие значения весов не допускают ослабления входного сигнала при прямом и обратном распространении.

С другой стороны, использование больших значений связано с серьёзными трудностями. Слишком большие значения весов могут привести к гиперчувствительности нейронной сети, что особенно заметно на примере рекуррентных сетей. Также большие значения сумм в нейронах могут привести к насыщению функции активации, что приводит к потере градиента и невозможности дальнейшего обучения.

Выбор подходящего распределения случайных значений для весов стал предметом исследования многих специалистов в области нейронных сетей. Наиболее часто предлагаются распределения, ориентированные на сохранение дисперсии выходного сигнала и градиента слоя. Такие распределения должны учитывать характеристики слоя, такие как количество входных сигналов и количество нейронов. Например, в работе [4] предлагается распределение

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right) \tag{6.1}$$

где m - число входных сигналов слоя, n - число нейронов в слое (число выходных сигналов).

Использование распределения, параметры которого задаются автоматически исходя из конфигурации сети, позволяют уменьшить пространство поиска решения за счёт уменьшения числа гиперпараметров. Целесообразность использования конкретных распределений проверена в экспериментальном исследовании.

Общая проблема подобных распределений заключается в том, что при увеличении количества нейронов или входов слоя, отдельные веса становятся слишком малыми. Решением этой проблемы может быть разреженная инициализация [3], когда у каждого нейрона ненулевым значением инициализируется только k весовых коэффициентов. На практике такой подход требует аккуратного выбора

инициализируемых весов для того, чтобы избежать потери данных.

В ситуациях, когда это позволяют доступные вычислительные ресурсы, можно определять параметры распределения с помощью методов глобального поиска. На этом, в частности, основан метод мультистарта с отсечением [13], хорошо подходящий для обучения нейронных сетей относительно небольшого размера.

4. Экспериментальное исследование

Для сравнения качества работы отдельных способов инициализации весовых коэффициентов проведём запуски процесса обучения с различными алгоритмами инициализации и различными параметрами на зафиксированных архитектурах нейронных сетей для следующих задач:

- распознавание рукописных цифр из набора данных MNIST [6] с помощью полносвязной и свёрточной сети;
- распознавание изображений из набора данных Fashion-MNIST [7] с помощью свёрточной нейронной сети;
- распознавание цифр с номеров домов из набора данных SVHN [8] с помощью свёрточной сети.

Набор данных MNIST представляет собой набор чёрно-белых изображений размером 28x28 пикселей, содержащих по одной белой цифре на чёрном фоне. Набор Fashion MNIST также состоит из чёрно-белых изображений размером 28x28 пикселей, содержащих фотографии предметов одежды и аксессуаров, разбитые на 10 классов. Набор SVHN состоит из цветных изображений цифр с номеров домов размером 32x32 пикселей.

Гиперпараметры обучаемых сетей - архитектура сети, функция и параметры распределения значений весовых коэффициентов, шаг обучения (при использовании стохастического градиентного спуска для обучения) или отметка об использовании алгоритма AdaDelta.

В качестве точности сети в данной статье принимается доля правильно распознанных элементов валидационной выборки.

Сравнение методов инициализации производится по средней точности, достигаемой нейронной сетью на последней эпохе, а также по минимальному количеству эпох, требуемых для достижения максимальной точности.

Для каждого набора гиперпараметров обучение сети повторялось 5 раз с переинициализацией начальных значений весов. Каждая сеть

обучалась фиксированное количество эпох: полносвязная сеть для распознавания изображений MNIST обучалась 50 эпох, свёрточная сеть для распознавания MNIST - 15 эпох, свёрточная сеть для распознавания Fashion MNIST - 20 эпох, для распознавания SVHN - 20 эпох.

В первом эксперименте сети инициализировались случайными значениями из нормального и равномерного распределений с разными параметрами. Цель эксперимента - установить зависимость качества обученной сети от параметров распределений. Для обучения сетей использовался адаптивный алгоритм AdaDelta [10].

Для каждой сети и типа распределения сравнивается средняя точность, а также минимальное количество эпох обучения, за которое была достигнута максимальная точность.

Результаты экспериментов представлены в виде тепловых карт. Для нормальных распределений по горизонтали изменяется математическое ожидание (по логарифмической шкале от 0 в положительную и отрицательную стороны), по вертикали - дисперсия (по логарифмической шкале). На второй карте - минимальное количество эпох, потребовавшееся для достижения максимальной точности (меньше - лучше, в случае, если соответствующее значение точности приемлемо).

В ходе экспериментов было выяснено, что форма «пятна» на тепловой карте для разных сетей различается. Часто максимальные значения точности сети наблюдаются в точке, где математическое ожидание близко к 0 и при относительно малых значениях дисперсии, но в то же время высокие значения наблюдаются и для распределений, смещённых относительно нуля. Привести результаты всех запусков в рамках одной статьи затруднительно, поэтому в качестве примера на рисунках 1, 2 приводятся результаты эксперимента для набора данных Fashion MNIST, на рисунках 3, 4 - для SVHN. Для набора данных MNIST тепловая карта в рамках данной статьи не приводится, но она похожа на таковую для Fashion MNIST.

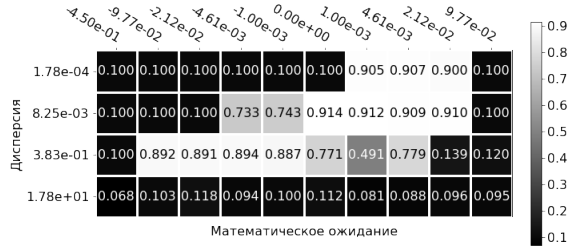

Рис. 1. Средняя точность сети на 20 эпохе, Fashion MNIST,

нормальное распределение.

Рис. 2. Число эпох для достижения максимальной точности, Fashion MNIST, нормальное распределение.

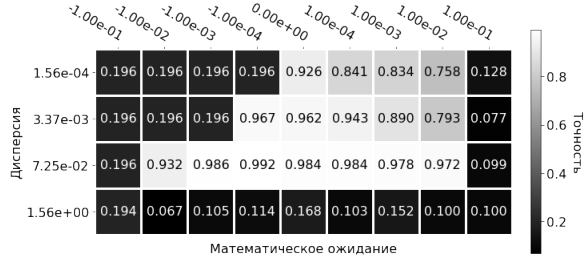

Рис. 3. Средняя точность сети на 20 эпохе, SVHN, нормальное распределение.

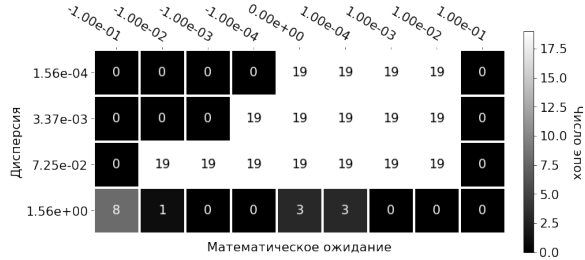

Рис. 4. Число эпох для достижения максимальной точности, SVHN, нормальное распределение.

Вторым этапом исследования является сравнение качества обученных нейронных сетей, для инициализации весов которых использовались часто используемые на практике функции распределения:

- `glorot_normal` [4] - нормальное распределение с математическим ожиданием 0 и дисперсией

$$D = \sqrt{\frac{2}{m+n}}$$

, где m - число входных сигналов слоя, n - число нейронов в слое;

- `glorot_uniform` [4] - равномерное распределение на интервале

$$\left[-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right]$$

.

Также для сравнения были выбраны следующие способы инициализации:

- `random_normal` - нормальное распределение с параметрами, соответствующими максимальной точности среди результатов первого эксперимента;
- `random_uniform` - равномерное распределение с параметрами, соответствующими максимальной точности среди результатов первого эксперимента;
- `orthogonal` - инициализация весовых коэффициентов ортогональными векторами [5].

Для обучения нейронных сетей используется метод стохастического градиентного спуска с различными фиксированными шагами обучения, а также алгоритм с адаптивной настройкой шага обучения AdaDelta [10].

В качестве примера на рисунке 5 изображена диаграмма зависимости средней точности сети от типа распределения и алгоритма обучения для свёрточной сети, обученной для распознавания изображений из набора Fashion MNIST, на рисунке 6 - из набора SVHN.

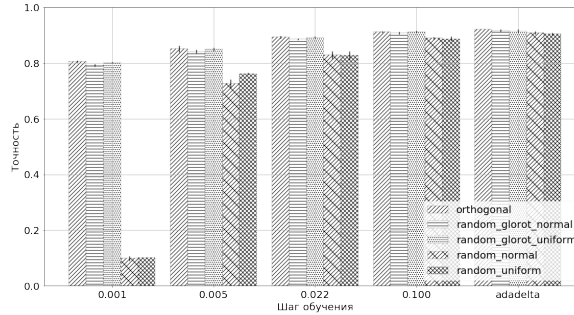

Рис. 5. Качество полученной сети в зависимости от шага обучения и метода инициализации, Fashion MNIST, свёрточная сеть.

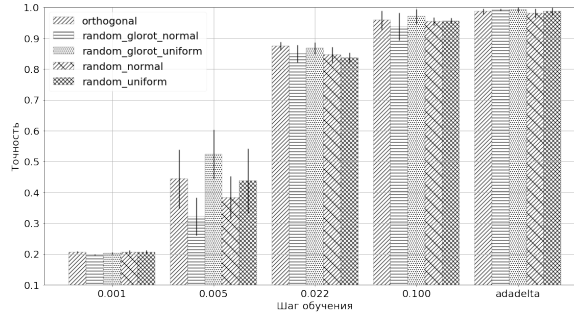

Рис. 6. Качество полученной сети в зависимости от шага обучения и метода инициализации, SVHN, свёрточная сеть.

В ходе экспериментов была продемонстрирована зависимость между параметрами «хорошего» распределения и конкретной задачей (сочетанием архитектуры сети, обучающей выборки и параметров алгоритма обучения). На тепловых картах видно, что для разных сетей максимальные значения качества наблюдаются при различных значениях параметров. Однако, наблюдается закономерность в расположении и форме областей с высоким качеством сетей: часто среднее значение весов близко к 0 и величина дисперсии относительно невелика.

Выбор заведомо «хорошего» распределения для начальной инициализации весов сети является нетривиальной задачей и требует специального исследования. Проведённые в рамках данной работы эксперименты позволили найти параметры распределений для нескольких задач, при которых в результате обучения получаются

качественные сети. Однако, на построение одной карты для нормального распределения и относительно простой свёрточной сети потребовалось 3.5 часа вычислений на GPU при времени на обучение одной сети, близкому к 1 минуте.

Использование методов инициализации, описанных в работах [4 и [5]], позволило во всех случаях получить очень высокие результаты качества работы сети за небольшое количество попыток. Часто результат, полученный с помощью этих методов, оказывался лучше полученного в результате исследования пространства распределений.

Использование адаптивных методов позволяет ускорить получение сети с требуемыми характеристиками за счёт сокращения количества настраиваемых параметров и, как следствие, упрощения и ускорения подготовки к обучению.

Заключение

Исследование, проведённое в рамках данной статьи, показало, что для разных задач параметры «хороших» приближений могут существенно различаться. Прямой поиск таких приближений требует много времени. При этом специальные методы инициализации, предложенные в работах [4] и [5], достаточно универсальны и приводят к хорошим решениям для разных классов задач.

Обучение нейронной сети - итеративный процесс, и вычислительная сложность зависит от многих факторов [14]. Выбор неудачного начального приближения может привести к тому, что процесс обучения не будет сходиться, в то же время «удачные» позволяют сократить количество эпох обучения, требуемых для достижения достаточной точности работы.

Таким образом, на основании проведённого исследования и результатов, полученных в работах [4] и [5], можно сделать вывод, что в общем случае допустимо использование описанных выше методов инициализации. Поиск альтернативных методов может иметь смысл в отдельных случаях и требует дополнительных исследований.

Литература

1. LeCun Y. [и др.]. *Backpropagation applied to handwritten zip code recognition.* Neural computation. – 1989. Т.1, №4.

2. Yam J. Y., Chow T. W. *A weight initialization method for improving training speed in feedforward neural network.* Neurocomputing. – 2000. T.30, №1–4.

3. Martens J. *Deep learning via Hessian-free optimization.* ICML. T.27 – 2010.

4. Glorot X., Bengio Y. *Understanding the difficulty of training deep feedforward neural networks.* Proceedings of the thirteenth international conference on artificial intelligence and statistics. – 2010.

5. Saxe A. M., McClelland J. L., Ganguli S. *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks.* arXiv preprint arXiv:1312.6120. – 2013.

6. LeCun Y. *The MNIST database of handwritten digits.* http://yann. lecun. com/exdb/mnist/. – 1998.

7. Xiao H., Rasul K., Vollgraf R. *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.* arXiv preprint arXiv:1708.07747. – 2017.

8. Netzer Y., Wang T. *Reading Digits in Natural Images with Unsupervised Feature Learning.* http://ufldl. stanford. edu/housenumbers. – 2011.

9. Bottou L. *Online learning and stochastic approximations.* On-line learning in neural networks. – 1998. T.17, №9.

10. Zeiler M. D. *ADADELTA: an adaptive learning rate method.* arXiv preprint arXiv:1212.5701. – 2012.

11. Cybenko G. *Approximation by superpositions of a sigmoidal function.* Mathematics of control, signals and systems. – 1989. T.2, №4.

12. Goodfellow I. [и др.]. *Deep learning.* T.1. – MIT press Cambridge, 2016.

13. Костенко В. А., Смолик А. В. *Алгоритм мультистарта с отсечением для обучения нейронных сетей прямого распространения.* Нейроинформатика-2007: Сборник научных трудов. – 2007. – Т.3.

14. Orponen P. *Computational complexity of neural networks: a survey.* Nordic Journal of Computing.– 1994.– Т. 1. №1.

Петров И.С., Шендяпин А.С.

Обзор средств обеспечения анонимности в SDN

Введение

Согласно терминологии Пфицмана и Хансена, для обеспечения анонимности система должна обладать свойством несвязываемости между отправителем и получателем, что означает отсутствие возможности определить точно их сетевое расположение для третьей стороны.

В наши дни обеспечение анонимности в интернете представляет собой крайне важную и актуальную задачу, стоящую как перед пользователями, так и перед провайдерами сервисов анонимности. В современных сетях существует множество недобросовестных провайдеров и злоумышленников, чьей целью является раскрытие сетевого расположения пользователя, и использование данных о его сетевой активности в своих целях.

Существует несколько сервисов, предоставляющих анонимность в сети. К первым следует отнести различные прокси и VPN серверы, позволяющие заменить IP-адрес пользователя, указываемый в пакетах, IP-адресом сервера. Второй подход основан на луковой маршрутизации — способе анонимной передачи многократно зашифрованных пакетов через несколько маршрутизаторов — и всех решениях, базирующихся на нем, таких, как Tor и mix-net. Еще одним способом достижения анонимности является использование I2P (англ. Invisible internet project) — устойчивой и анонимной сети, работающей поверх сети Интернет.

Однако все перечисленные системы обладают рядом недостатков, которые будут подробно рассмотрены далее, одним из ключевых среди которых является производительность.

Облачные вычисления — это модель предоставления удаленного доступа к вычислительным ресурсам. К таким ресурсам могут быть отнесены центры хранения и обработки данных, распределенные сети передачи информации, различные приложения и сервисы. Основным достоинством данного подхода являются сокращение нагрузки на устройства пользователей и гибкость управления сервисами.

SDN (англ. Software defined networking) — это сетевая архитектура, основная идея которой заключается в разделении уровня передачи данных и уровня управления, который выносится на отдельный элемент подсети, называемый контроллером. Главной функци-

ей контроллера является управление маршрутизацией в вверенной подсети — установка правил по обработке потоков и взаимодействие с приложениями пользователей. Функцией коммутаторов является передача пакетов согласно установленным правилам.

Применение этих моделей в рамках сервисов анонимности позволяет решить проблемы существующих средств и предложить новые способы обеспечения анонимности пользователей, применение которых ранее было невозможным из-за децентрализованного управления подсетями.

В данной статье проводится обзор существующих средств предоставления анонимности при помощи облачных платформ и архитектуры SDN. Системы сравниваются на основе выбранных признаков, описывающих основные характеристики таких систем. В статье также выделены основные достоинства и недостатки существующих систем.

Статья организована следующим образом. В первой главе описываются основные критерии сравнения сервисов анонимности. Во второй главе проводится обзор систем обеспечения анонимности. В третьей главе описаны результаты сравнения рассмотренных систем. В заключении сформированы основные выводы по теме.

1. Критерии сравнения

Защита анонимности. Главной задачей сервисов анонимности является защита анонимности пользователя. Сеть передачи данных почти всегда считается ненадежной и потенциально может быть скомпрометирована. Если устройства передачи данных будут знать сетевое расположение пользователя и ресурса, с которым он связывается, то для раскрытия личности пользователя будет достаточно захвата злоумышленником хотя бы одного из них;

Производительность. Одной из главных проблем современных сервисов обеспечения анонимности является их низкая производительность. Решение данной проблемы позволило бы применять сервисы анонимности в более широкой области сетевых коммуникаций, требующей малой задержки доставки пакетов, например, IP-телефонии и IP-телевидении;

Балансировка между обеспечением нужных пользователю уровней анонимности и производительности, предоставляемых сервисом. Большинство из рассмотренных инструментов являются недостаточно гибкими в этом отношении. В ряде случаев пользователю может потребоваться увеличить производительность системы, ослабив анонимность подключения. В некоторых случа-

ях, наоборот, приоритетной целью является уровень обеспечиваемой анонимности. Реализация возможности смещения приоритетов позволяет привлечь пользователей, для которых переключение между этими параметрами является важным приоритетом при выборе сервиса;

Анонимность по отношению к провайдеру. Еще одна важная проблема некоторых существующих сервисов (различных прокси-серверов, VPN, SSH туннелей) состоит в необходимости доверять провайдеру сервиса. То есть при использовании пользователями скомпрометированного сервиса злоумышленник получает доступ к информации об установленных соединениях, которую может затем использовать в своих целях.

2. Обзор систем обеспечения анонимности

2.1 Традиционные сети

К одним из основных сервисов анонимности можно отнести различные прокси и VPN серверы, которые обеспечивают анонимность за счет передачи трафика на сервер, который заменяет IP-адрес пользователя в принятых пакетах на свой собственный, и затем пересылает пакеты на сетевой ресурс. Данный способ достаточно прост в реализации, однако требует доверия к серверу, поскольку последний знает и IP-адреса клиента и сетевого ресурса, к которому клиент обращается.

Другим решением является метод луковой маршрутизации, разработанный в 1990-х годах [1], и основанные на нем инструмент Tor и стек протоколов mix-net [2] [3] [4]. Его основной идеей является построение цепочки маршрутизаторов, каждый из которых обладает информацией только об одном участке сетевого подключения — между ним и следующим маршрутизатором в цепочке. Конфиденциальность обеспечивается с помощью согласования пользователем индивидуальных симметричных ключей шифрования с каждым из маршрутизаторов.

При отправке сообщения в пакете указывается маршрутизирующая информация для последнего маршрутизатора в цепочке, пакет шифруется и сохраняется как поле данных другого пакета, в качестве маршрутизирующей информации для которого указывается информация о предпоследнем участке соединения, и так далее. При получении пакета, каждый маршрутизатор, используя свой ключ

шифрования, снимает верхний слой шифрования и передает пакет дальше.

Благодаря тому, что ни один маршрутизатор не знает одновременно оба IP-адреса — отправителя и получателя, даже при компрометации нескольких из них анонимное соединение может быть построено.

Однако в случае, если между маршрутизаторами в сети присутствуют другие сетевые устройства, через которые проходят пакеты и к которым может иметь доступ злоумышленник, данную защиту можно обойти с помощью таймингого анализа трафика [2], заключающегося в сопоставлении пакетов, входящих и исходящих с сетевого узла.

Возможное решение данной проблемы предлагает система mix-net, основанная на луковой маршрутизации и одноименном стеке протоколов. Маршрутизация осуществляется, как и в Tor, но с одним дополнением: все пришедшие за определенный период времени пакеты сохраняются на маршрутизаторе и по истечении времени ожидания пересылаются на следующие узлы в случайном порядке. Данный метод устойчив к анализу трафика, но существенно увеличивает задержку на доставку пакетов.

Данная проблема частично решается в работе [5]. Ее основная идея заключается во введении порогового значения N, и дальнейшем изменении этого значения и времени ожидания T за счет событий, произошедших в сети. В случае, если число пришедших на маршрутизатор пакетов превышает N, они все рассылаются на их адреса назначения, в противном случае с некоторой вероятностью они пересылаются на другие вершины, а пороговое значение уменьшается. Время ожидания T вычисляется на каждом шаге с помощью алгоритма имитации отжига.

Tor и mix-net предоставляют хороший уровень анонимности, в том числе по отношению к маршрутизаторам, однако в качестве их главного недостатка следует выделить низкую производительность.

I2P (англ. Invisible internet project) — анонимная сеть, работающая поверх сети Интернет. Для анонимизации отправляемых сообщений каждое приложение-клиент имеет свой I2P "маршрутизатор строящий несколько туннелей для входящего и исходящего трафика. Установка соединения между клиентами осуществляется за счет обращения маршрутизаторов к базе данных, расположенной на удаленном сервере, и предоставляющей списки туннелей в сети.

Еще одним способом обеспечения анонимности является подход проактивной изменяемой защиты [6]. Он базируется на замене деанонимизирующей пользователя маршрутизирующей информации в

заголовках пакетов периодически обновляемыми виртуальными адресами или идентификаторами — сущностями, также обеспечивающими маршрутизацию, но обладающими своей структурой. Данный метод стал возможен благодаря парадигме SDN, предоставляющей централизованное управление сетью, поскольку для корректной передачи пакетов требуется одновременное обновление правил на нескольких или даже на всех таблицах маршрутизации.

2.2 Облачные сервисы

Одним из методов повышения производительности сервисов предоставления анонимности является использование облачной инфраструктуры (англ. Cloud onion routing, COR), и использовании подхода разделения доверия [7].

Его основной целью является защита анонимности пользователя от провайдеров сервисов анонимности и провайдеров сетевой инфраструктуры.

Для установки анонимного соединения пользователь итеративно строит туннель через цепочку маршрутизаторов-ретрансляторов, как и в Tor, но принадлежащих разным сетевым провайдерам. Число провайдеров облачной инфраструктуры ограничено, однако за счет скорости передачи пакетов в подсетях, подконтрольных одному провайдеру, повышается производительность всей системы в целом.

Однако при данном подходе перед пользователем возникает несколько проблем: 1) За доступ к ретрансляторам необходимо платить, и, поскольку на этом этапе пользователь не имеет доступа к сервисам анонимности, операция оплаты может раскрыть личность пользователя; 2) Необходимо строить сетевое подключение таким образом, чтобы сохранять анонимность по отношению как к провайдерам сети, так и к провайдерам анонимности.

Первая проблема решается с помощью предоставления пользователю доступа к сети начальной загрузки. Она функционирует следующим образом: любой пользователь может запросить у провайдера сервиса анонимности ретранслятор для построения соединения для оплаты токенов. Присоединив полученный ретранслятор к цепочке, он через него запрашивает ретранслятор у другого провайдера, и так далее. Построенное соединение не раскрывает личность пользователя, поскольку идентично соединению, построенному по протоколу Tor.

Однако как построение анонимного соединения, так и построение цепочки ретрансляторов в сети начальной загрузки требует соблюдения ряда правил для защиты от провайдеров сетевой инфраструктуры и анонимных сервисов:

1. Входные и выходные вершины построенных через подсети цепочек должны различаться. Это условие обеспечивает защиту от таймингого анализа трафика, описанного в разделе 2.1. При выполнении данного правила злоумышленнику вне подсети будет гораздо труднее сопоставить входящий и исходящий трафик. Кроме того, эта атака усложняется также тем, что подсети, как правило, имеют несколько точек входа и выхода, и злоумышленнику приходится прослушивать их все.

2. Входная и выходная вершины построенного туннеля должны принадлежать разным провайдерам сетевой инфраструктуры и сервисов анонимности. В противном случае IP-адреса отправителя и получателя могут быть раскрыты этим провайдером.

3. Между цепочками ретрансляторов, предоставляемых одним провайдером облачной инфраструктуры или сервиса анонимности, не должно быть цепочек ретрансляторов других провайдеров, поскольку они не предоставляют дополнительной анонимности.

Существует также аналогичное решение для переноса механизма луковой маршрутизации в SDN (SOR) [8]. Системы, реализующие оба этих варианта, имеют идентичную структуру, которая в этой работе была описана относительно облачной платформы.

2.3 SDN

Парадигма SDN позволяет реализовать еще один способ защиты анонимности в сетях, основанный на механизме проактивной изменяемой защиты (англ. Moving Target Defense, MTD).

Суть данного способа состоит в замене маршрутизирующей информации, указываемой в пакетах, виртуальными, периодически обновлемыми IP-адресами или идентификаторами. Главной задачей такого подхода является усиление защиты анонимности соединения, в том числе — от таймингого анализа трафика. Оно достигается за счет периодического обновления виртуальных IP-адресов или идентификаторов, которое приводит к существенному усложнению сопоставления входящего и исходящего трафика на коммутаторах.

Однако при обновлении идентификаторов необходимо также обновлять соответствующие правила в таблицах маршрутизации. Для этой операции требуется централизованное управление коммутаторами, которое предоставляет парадигма SDN.

Существует несколько реализаций данного подхода, такие как PHEAR [9], усовершенствованный с помощью полуструктурирова-

ния идентификаторов [10] [11], различные методы замены IP-адресов [6]. Однако их архитектура является идентичной: сетевая инфраструктура представлена в виде SDN коммутаторов, управляемых SDN контроллером. На нем как приложение работает сервер провайдера сервиса анонимности, а на конечных пользовательских устройствах запущены локальные прокси. Функции каждой компоненты будут описаны ниже.

В данной модели SDN коммутаторы считаются потенциально скомпрометированными, в то время как SDN контроллер, работающий на нем сервер и локальные прокси полагаются защищенными.

В качестве реализации парадигмы SDN в предложенных решениях используется протокол OpenFlow. Поскольку сервис вмешивается в работу других приложений, на коммутаторах необходимо хранить до четырех таблиц:

1. Входная таблица. Ее функция — классификация пакетов, то есть определение последовательности остальных таблиц, которые должен пройти каждый пришедший пакет.

2. Таблица отправляемых пакетов. Необходима только на граничных коммутаторах. Служит для преобразования маршрутизирующей информации.

3. Таблица маршрутизации. Ее назначение — маршрутизация по таблице потоков в сети.

4. Таблица принимаемых пакетов. Необходима только на граничных коммутаторах. Служит для обратного преобразования маршрутизирующей информации.

В ряде решений, основанных на изменении IP-адреса [6], [12] для обеспечения прозрачности системы по отношению к другим приложениям используется база данных соответствий реальных и виртуальных IP-адресов, хранящаяся на контроллере.

Сервер работает на SDN контроллере как приложение и имеет три функции — построение таблиц маршрутизации, реакция на сообщения прокси и обновление идентификаторов коммутаторов (IP-адресов). Таблицы строятся с учетом топологии — сервер строит ее с помощью протокола OFDP.

Запросы прокси делятся на 3 вида:

1. Регистрация хостов.

2. ARP запрос.

3. Запрос на обновление идентификатора (IP-адреса) хоста.

Локальные прокси расположены на границе области доверия — они выполняют роль посредника между хостами пользователей и сетью SDN коммутаторов. Их основной функцией является перевод IP и MAC-адресов хоста пользователя в виртуальную маршрутизирующую информацию, и наоборот.

Также помимо обновления только IP и MAC-адресов в работах [10] и [12] рассмотрены возможности изменения портов, достигаемые за счет пре- и постпроцессинга портов, выполняющего преобразование номеров портов из случайных в реальные и наоборот, и перестройки сетевого подключения — для этого достаточно перед запуском сети с помощью поиска в ширину найти всевозможные маршруты сетевых соединений, и затем обновлять его с заданной периодичностью или вероятностью.

3. Сравнение сервисов

Сравнение рассмотренных средств обеспечения анонимности осуществляется на основе описанных в разделе 1 критериев: Защиты анонимности, обеспечения производительности и балансировки между ними, а также необходимостью доверять провайдеру.

Поскольку средства, использующие методы проактивной защиты имеют одинаковую архитектуру, они представлены с таблице в сувокупности.

Различия между средствами обеспечения анонимности могут быть в первую очередь объяснены различием их основных целей. Так, например, главной задачей подходов COR и SOR, базирующегося на переносе луковой маршрутизации в облачную и SDN инфраструктуру, является обеспечение анонимности по отношению к провайдеру и достижение большей производительности, нежели в Tor. С другой стороны, сервисы, использующие методы проактивной защиты, концентрируются в первую очередь на защите анонимности подключения от злоумышленника извне сервиса, не рассматривая необходимость обеспечивать анонимность по отношению к провайдеру.

Таким образом, использование того или иного сервиса зависит от конкретных нужд пользователя, поскольку различные сервисы ставят перед собой и решают разные задачи.

Сервис анонимности	Степень защищенности	Производительность	Балансировка	Анонимность для провайдера
Прокси-серверы	Низкая	Высокая	-	-
VPN-серверы	Высокая	Высокая	-	-
Tor	Высокая	Низкая	-	+
I2P	Высокая	Низкая	-	+
COR, SOR	Высокая	Средняя	+	+
Методы проактивной защиты	Крайне высокая	Высокая	+	-

Таблица 1: Сравнение сервисов

Заключение

Несмотря на то, что проблема обеспечения анонимности в сети была впервые обозначена в 1980-е годы, она по-прежнему сохраняет свою актуальность.

В данной работе были рассмотрены существующие сервисы анонимности и их основные критерии: защита анонимности, повышение производительности, балансировка между этими параметрами и обеспечение анонимности по отношению к провайдеру сервиса.

Были описаны два основных пути, по которым развиваются сервисы анонимности — разделение доверия между провайдерами, имеющее основной целью защиту анонимности по отношению к провайдеру сервисов и повышение производительности, и технология проактивной изменяемой защиты, главной целью которой является защита анонимности от злоумышленников извне.

Рассмотренные подходы, как следует из их оценок на основе описанных критериев, могут дополнять друг друга, и в будущем могут быть сделаны шаги по их объединению. Также перспективным направлением исследования является снятие допущения о добросовестности SDN контроллера при маршрутизации в одной сети.

Литература

1. Goldschlag D., Reed M., Syverson P. Onion routing for anonymous and private Internet connections. // Communications of the ACM 42(2). 1999. P. 39–41.

2. Edman M., Yener B. On anonymity in an electronic society: A survey of anonymous communication systems. // ACM Computing Survey, vol. 42, N 1. 2009.

3. Danezis G., Diaz C. A survey of anonymous communication channels // Technical Report MSR-TR-2008-35, Microsoft Research. 2008.

4. Ren J., Wu J. Survey on anonymous communications in computer networks. // Comput. Commun., vol. 33. N 4. 2010. P. 420–431.

5. Shen Z., T., M., Wang M., Zhu L., and Li F. Self-adaptive anonymous communication scheme under SDN architecture // In Computing and Communications Conference (IPCCC). 2015 IEEE 34th International Performance. P. 1–8.

6. Jafarian J. H., Al-Shaer E., & Duan Q. Openflow random host mutation. // Proceedings of the First Workshop on Hot Topics in Software Defined Networks — HotSDN '12.

7. Jones N., Arye M., Cesareo J., and Freedman M. J. Hiding amongst the clouds: A proposal for cloud-based onion routing. // In FOCI. 2011.

8. Elgzil A., Chow C. E., Aljaedi A., & Alamri N. Cyber anonymity based on software-defined networking and Onion Routing (SOR) // 2017 IEEE Conference on Dependable and Secure Computing.

9. Skowyra R., Bauer K., Dedhia V., Okhravi H. Have no phear: Networks without identifers. // Proceedings of the 2016 ACM Workshop on Moving Target Defense, MTD '16, ACM: New York, NY, USA, 2016. P. 3–14.

10. Wang Y., Yi J., Guo J., Qiao Y., Qi M., & Chen Q. A Semistructured Random Identifier Protocol for Anonymous Communication in SDN Network. // Security and Communication Networks. 2018. P. 1–20.

11. Jain S., Chen Y., Zhang ZL., Jain S. Viro: A scalable, robust and namespace independent virtual id routing for future networks. // 2011 Proceedings IEEE INFOCOM, 2011. P. 2381–2389

12. Chavez A. R., Stout W. M. S., & Peisert S. Techniques for the dynamic randomization of network attributes. // 2015 International Carnahan Conference on Security Technology (ICCST)

Синякова М.А., Степанов Е.П.

Анализатор состояния компьютерной сети на основе теории стохастического сетевого исчисления[1]

Введение

Существует ряд задач сетевой инженерии, для которых необходимо уметь прогнозировать задержки и потери пакетов при передаче трафика через сеть. Указанные характеристики критичны для задач передачи голосового и видео трафика, обмена информацией между биржевыми сервисами и банками. Также они важны, например, при выборе оптимального маршрута передачи данных через сеть. Для оценки задержки и заполненности буферов сетевых устройств широко используется сетевое исчисление. Существуют два вида сетевого исчисления: детерминированное [1] и стохастическое [2]. Первый вид в основном использует кусочно-линейные функции для оценки поступления трафика в сеть, который учитывает все пики и всплески трафика, что дает верхнюю оценку для задержки и отставания, которая никогда не может быть нарушена. Стохастический вид сетевого исчисления отбрасывает маловероятные резкие всплески трафика, что позволяет промоделировать работу сети в среднем, а не в наихудшем случае, из этого получаем оценку задержки и отставания с определенной вероятностью. Один из недостатков детерминированного сетевого исчисления заключается в том, что его оценки слишком консервативны - в большинстве состояний системы оцениваемые величины не достигают своих наибольших значений. Еще один недостаток детерминированного сетевого исчисления заключается в ограниченном классе функций прибытия пакетов, к которым можно применить данный метод, так как кривые нагрузки представимы в основном кусочнолинейными или кусочно-непрерывными функциями. Таким образом, моделирование реального состояния загруженности сети затруднено [3][4].

При помощи стохастического представления трафика и работы передающих устройств становится возможным более точно промоделировать работу сети [4][5], а также расширить класс функций прибытия пакетов, к которым можно применить данный метод сетевого исчисления, на более сложные модели работы сети. В настоящее

[1]Работа выполнена при частичной поддержке РФФИ, грант № 18-07-01255

время стохастический метод уже используется при проектировании программно-конфигурируемых сетей [6], например, при определении оптимального положения управляющего контроллера. Кроме того, оценки задержки, получаемые при помощи стохастического сетевого исчисления, могут использоваться при разработке распределенных приложений под конкретную сеть, что упростит настройку сети и поможет в ее оптимизации.

1. Введение в стохастическое сетевое исчисление

В данной работе мы хотим использовать стохастический аппарат сетевого исчисления и получить ограничения для задержки и отставания, которые были бы наиболее приближены к реальной работе сети. Введем основные определения:

- *Функция прибытия* $A(t)$ - описывает зависимость суммарного количества данных, поступивших на обработчик, от времени.
- *Функция обработчика* $S(t)$ – зависимость количества переданных данных от времени.
- *Отставание (backlog)* $B(t)$ – выражает количество данных, находящихся «внутри» обработчика.
- *Задержка (delay)* $W(t)$ – время прохождения через обработчик той порции данных, которая поступила на него в момент времени t.

Представим ограничение для отставания в виде $P[B(t) > b] \leq \varepsilon'$, где ε' вероятность того, что в реальной сети отставание нарушит построенную оценку b. Для задержки мы хотим получить ограничение в виде $P[W(t) > \omega] \leq \varepsilon'$, где ε' вероятность того, что в реальной сети задержка нарушит построенную нами оценку ω. Данные оценки можно получить из моделей EBB (exponentially bounded burstiness) и EBF (exponentially bounded functions). Рассмотрим модель EBB на примере входного трафика для фиксированного значения τ:

$$P[A(\tau, t) > \rho(t - \tau) + b] \leq \varepsilon(b), \quad \varepsilon(b) = \alpha e^{-\theta b} \tag{1.1}$$

где $A(\tau, t)$ – входной трафик, пришедший в момент времени от τ до t ; ρ – скорость прибытия пакетов, которая зависит от свободного параметра $\theta \geq 0$; b – размер всплеска; $\varepsilon(b)$ – функция, обозначающая вероятность, с которой допускается нарушение построенной оценки

для случайной величины $A(\tau, t)$ и зависит от параметра b, коэффициент $\alpha \geq 0$. Модель называется EBB так как при увеличении b вероятность нарушения оценки экспоненциально уменьшается.

Чтобы применить соотношения, справедливые для детерминированного сетевого исчисления, для получения оценки задержки, необходимо, чтобы можно было взять любое $\tau \in [0, t]$. Поэтому обобщение неравенства (1.1) для всех значений τ представимо в виде:

$$P[\exists \tau \in [0,t] : \ A(\tau, t) \ > \ \rho'_A(t-\tau) + b_A] \ \leq \ \varepsilon'(b_A), \qquad (1.2)$$

где $\rho'_A = \rho_A + \delta$, а δ – калибровочный параметр, $\varepsilon'(b_A)$ – вероятность, с которой может нарушаться построенная оценка. Величины ρ_A и b_A характеристики трафика.

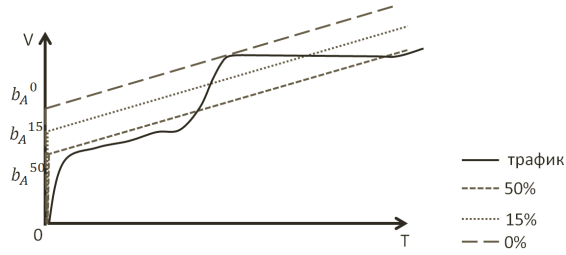

Рис. 1: Графическое представление зависимости вероятности в модели EBB от величины b_A.

Для обработчика применима модель EBF (exponentially bounded functions), которая аналогична модели EBB и имеет вид:

$$P[S(\tau, t) < \rho(t-\tau) - b] \ \leq \ \varepsilon(b), \quad \varepsilon(b) \ = \ \alpha e^{-\theta b} \qquad (1.3)$$

$$P[\exists \tau \in [0,t] : \ S(\tau, t) \ < \ \rho'_S(t-\tau) + b_S] \ \leq \ \varepsilon'(b_S), \qquad (1.4)$$

Исходя из утверждений (1.2) и (1.4) получим диапазон выбора калибровочного параметра и основное ограничение на скорости трафика и обработчика. Мы вводили обозначения как $\rho'_A = \rho_A + \delta$ и $\rho'_S = \rho_S - \delta$.

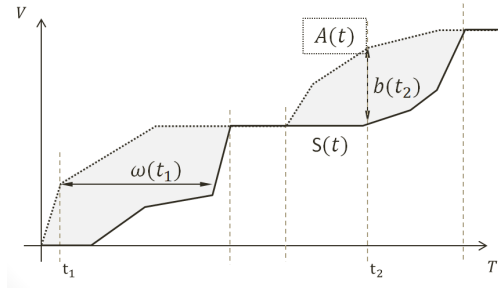

Рис. 2: Визуализация значений задержки и отставания.

На Рис.2 отставание – максимальное отклонение между двумя прямыми по вертикали, а задержка – максимальное отклонение между двумя прямыми по горизонтали. Так как обе оценочные функции имеют вид кусочно-линейных функций, то нам для получения конечных оценок отставания и задержки необходимо пересечение данных прямых или сохранение параллельности, то есть необходимо выполнение условия $\rho_A \leq \rho_S$. Исходя из необходимого условия выполнения оценок, получаем диапазон для выбора δ, $\delta \in [0, (\rho_A - \rho_S)/2]$.

Модели EBB и EBF хороши для расчетов, но в большинстве случаев, такие оценки получить трудно, так как случайные величины поступление трафика и работа обработчиков в основном задаются распределением. Для работы с таким представлением случайной величины введем модель MGF (moment generation function) – производящая функция моментов.

Производящей функцией моментов (MGF) называется функция от случайной величины X с распределением $\mathbb{F}^X$, имеющая вид: $M_X(\theta) = \mathbb{E}[e^{\theta X}]$, где θ – случайный параметр.

Введем аффинную оценку для MGF трафика:

$$\mathbb{E}[e^{\theta A(\tau,t)}] \ = \ e^{\theta(\rho(t-\tau)+\sigma)} \tag{1.5}$$

Аффинная оценка MGF функции обслуживания:

$$\mathbb{E}[e^{-\theta S(\tau,t)}] \ \leq \ e^{-\theta(\rho(t-\tau)-\sigma)} \tag{1.6}$$

Используя неравенство Чернова $P[X \geq x] \leq e^{-\theta x}\mathbb{E}[e^{\theta} X]$ можно из аффинной оценки MGF получить модели EBB и EBF для трафика и обработчика соответственно.

Модель MGF имеет ряд преимуществ, которые могут упростить выполнение некоторых задач:

- Мультиплексирование трафика
- Планирование трафика

Используя выше рассмотренные подходы, авторы статьи [5] рассчитали вероятности $\varepsilon'_A(b) = \frac{e^{\theta\sigma}}{1-e^{-\theta\delta}}e^{-\theta b}$ и $\varepsilon'_S(b) = \frac{e^{\theta\sigma}}{1-e^{-\theta\delta}}e^{-\theta b}$ для трафика и обработчика соответственно. $\varepsilon'(b) = \varepsilon'_A(b) + \varepsilon'_S(b)$ - это общая вероятность, с которой допускается нарушение полученной оценки для характеристик сети. Инвертированием формул вероятностей $\varepsilon'_A(b)$ и $\varepsilon'_S(b)$ получаем основные формулы для вычисления параметров b_A и b_S, которые в дальнейшем будут использованы для вычислений задержки и отставания в сети.

$$b_A \ = \ \sigma_A - \frac{1}{\theta}\left(\ln\frac{\varepsilon'}{2} + \ln\left(1 - e^{-\theta\delta}\right)\right) \tag{1.7}$$

$$b_S = \sigma_S - \frac{1}{\theta}\left(\ln\frac{\varepsilon'}{2} + \ln\left(1 - e^{-\theta\delta}\right)\right) \tag{1.8}$$

2. Модель сети

В качестве модели трафика используем Марковский трафик. В этой модели трафика процессы прибытия имеют память первого порядка, то есть текущее состояние зависит от предыдущего.
Имеется 2 состояния:

- В состоянии 1 трафик не поступает, режим Off;
- В состоянии 2 пакеты данных поступают с постоянной скоростью r, режим On.

Так как трафик отправляется только в одном из этих состояний, то нам необходимо выяснить, какую часть от всего времени работы составляет состояние отправки трафика. Для этого вычислим вероятность нахождения в состоянии On: $p_{on} = p_{12}/(p_{12} + p_{21})$, где p_{ij} это вероятности перехода из одного состояния в другое. Получаем среднюю скорость поступления трафика $p_{on}r$. Данная модель удовлетворяет аффинной оценки MGF (1.5) с параметрами $\sigma = 0$ и скоростью ρ, описанной в статье [5]. В частном случае, когда $p_{11} = p_{21}$ и $p_{22} = p_{12}$, то $p_{on} = p_{22}$,то есть важно в какое состояние мы перешли, и не важно, в каком находились до этого:

$$\rho = \frac{\ln\left(p_{on}e^{\theta r} + 1 - p_{on}\right)}{\theta} \tag{2.1}$$

Применим теорию стохастического сетевого исчисления к вычислению оценки отставания и задержки для примитивной модели сети, состоящей из одного узла. Данный пример был разобран авторами работы [5]. Сначала рассмотрим обработчик, работающий с постоянной скоростью c. В этом случае его функция обслуживания будет иметь вид $S(\tau, t) = \rho_S(t - \tau) - \sigma_S$, где $\sigma_S = 0$, а $\rho_S = c$. В представлении данного обработчика в модели EBF (1.4) калибровочный параметр $\delta = 0$. Так как сервер работает с постоянной скоростью, то для уменьшения вероятности, с которой допускается нарушение оценки EBF, не нужно уменьшать наклон кривой обслуживания и осуществлять сдвиг относительно начала координат на b_S. Для данного сервера $\varepsilon'_S(b_S) = 0$, так как $\rho'_S = \rho_S = c = const$. Теперь оценим функцию прибытия пакетов. В связи с тем, что $\varepsilon'_S(b_S) = 0$ формула (1.7) представима:

$$b_A = \sigma_A - \frac{1}{\theta}\left(\ln\varepsilon' + \ln\left(1 - e^{-\theta(c - \rho_A)}\right)\right). \tag{2.2}$$

Параметр δ выбираем в граничных точках интервала области определения. Данная формула зависит от случайной величины θ, график зависимости от которой приведен ниже на Рис. 3.

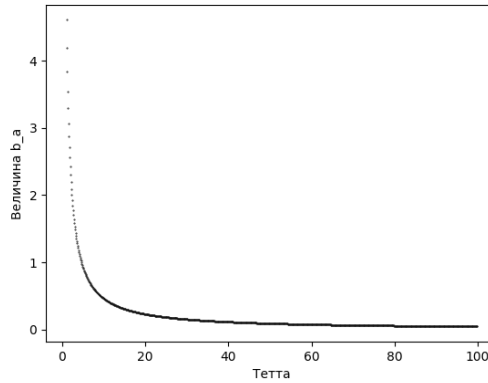

Рис. 3: График зависимости b_A от θ.

Тем самым, если считать ρ_A не зависимым от θ и при фиксированных остальных параметрах, а также при выполнении необходимого условия $c < \rho_A$ получаем, что $b_A \to \sigma_A$ при $\theta \to +\infty$.

Для того чтобы оценить работу сети, получим общие формулы для оценки задержки и отставания, считая, что кривая прибытия и кривая обслуживания имеют следующие оценки: $A(\tau,t) \leq \rho'_A(t-\tau)+b_A$, $S(\tau,t) \geq \rho'_S(t-\tau)-b_S$, где каждое из соотношений нарушается с вероятностями ε'_A и ε'_S соответсвенно. Также будем учитывать необходимое условие выполнения оценки $\rho_A \leq \rho_S$.
Оценить отставание $B(t)$ сверху можно при помощи выражения:

$$B(t) \leq \max_{\tau\in[0,t]}\{A(\tau,t) - S(\tau,t)\}. \tag{2.3}$$

Аналогичным образом найдем оценку сверху для задержки $W(t)$ из неравенства:

$$W(t) \leq \min_{\omega\geq 0}\{\max_{\tau\in[0,t]}\{A(\tau,t) - S(\tau,t+\omega)\} \leq 0\}. \tag{2.4}$$

Подставляя в формулы (2.3) и (2.4) ограничение на функцию трафика из (1.2) и на функцию обработчика из (1.4) мы получим формулы для оценки отставания и задержки:

$$b = b_A + b_S \times \frac{\rho_A + \delta}{\rho_S - \delta}, \quad \omega = \frac{b_A + b_S}{\rho_S - \delta}.$$

Так как мы рассматриваем случай с упрощенным обработчиком, то полученные величины будут выражаться формулами: отставание $b = b_A$, задержка $\omega = \frac{b_A}{c}$. Данные соотношения следуют из того, что $b_s = 0$ и $\delta = 0$.

Рассмотрим более сложный и реалистичный случай сервера с начальной разрывной мерой σ_S. Так как данный обработчик не является стохастическим, то для него калибровочный параметр $\delta = 0$ и вероятность нарушения данной оценки $\varepsilon'_S(b_S) = 0$. Тогда для такого обработчика и любого вида трафика будет справедлив набор формул:

$$b_A = \sigma_A - \frac{1}{\theta}(\ln \varepsilon' + \ln(1 - e^{-\theta(c - \rho_A)})).$$

$$b_S = \sigma_S,$$

$$b = b_A + b_S \times \frac{\rho_A + \delta}{\rho_S},$$

$$\omega = \frac{b_A + b_S}{\rho_S}.$$

В настоящей работе все исследования будут производиться на основе именно этих формул.

3. Экспериментальное исследование

В данном разделе сравним оценки, получаемые при помощи стохастического и детерминированного сетевого исчисления. Проведем экспериментальное исследование для Марковского трафика на примере линейной топологии сети, где в роли каждого обработчика выступает work-conserving server с постоянной скоростью работы c_i при $i \in [1, n]$, где n– количество обработчиков.

Был написан прототип на языке Python3, вычисляющий задержку и отставание для двух сетевых исчислений. Ввод данных в данный прототип осуществляется в формате json, в котором задается количество обработчиков, их параметры, информация о топологии сети, тип входного трафика и его параметры.

Для начала проводим свертку данных обработчиков [5] и получаем значения c и b_S - общая характеристика сети. Считаем, что после свертки все значения обраотчиков перешли в два параметра c и b_S. Рассмотрим соответствующие параметры для трафика. Так как трафик Марковский, то для него имеем параметры $\sigma_A = 0$ и $\rho_A = \frac{\ln(p_{on}e^{\theta r} + 1 - p_{on})}{\theta}$. Ограничение функции прибытия для Марковского трафика в виде линейной функции в детерминированном

сетевом исчислении будет иметь вид $\rho_A = r$ и $\sigma_A = 0$. Таким образом, в детерминированном сетевом исчислении при построении оценки не учитывается вероятность p_{on} и считается, что трафик приходит с постоянной скоростью r.

Теперь найдем значение b_A для стохастического исчисления. В формулу (2.2) подставим значение ρ_A для Марковского трафика – формулу (2.1), получаем представление:

$$b_A = \sigma_A - \frac{1}{\theta}(\ln(\varepsilon') + \ln(1 - e^{-\theta c}(p_{on}e^{\theta r} + 1 - p_{on}))). \quad (3.1)$$

Оценка b_A справедлива для любого допустимого значения параметра θ, поэтому можно улчушить значение оценки, если найти такое значение параметра θ, при котором оценка будет минимальна. Возьмем производную от (3.1) по θ, чтобы исследовать зависимость формул оставания и задержки от θ.

$$(b_A)' = \frac{1}{\theta^2}\left(\ln\varepsilon' + \ln\left(1 - e^{-\theta c}(p_{on}e^{\theta r} + 1 - p_{on})\right)\right) - \\ - \frac{1}{\theta}\frac{(c-r)p_{on}e^{\theta(c-r)} + ce^{-\theta c} - p_{on}ce^{-\theta c}}{1 - e^{-\theta c}(p_{on}e^{\theta r} + 1 - p_{on})}. \quad (3.2)$$

Оценим знак производной. Рассмотрим первое слагаемое. Оно на всей области значений θ принимает отрицательные значения, так как первый множитель $\frac{1}{\theta^2} \geq 0$ при всех значения θ. Второй множитель отрицательный в силу отрицательности обоих логарифмов. Так как $\varepsilon' \in [0,1]$ – вероятность нарушения построенной оценки, то $\ln(\varepsilon') < 0$. Так как $c \geq 0$ и из выполнения необходимого условия применимости метода $\rho_A \leq \rho_S$ следует неравенство:

$$p_{on}e^{\theta r} + 1 - p_{on} \leq e^{\theta c}, \quad (3.3)$$

то $\ln(1 - e^{-\theta c}(p_{on}e^{\theta r} + 1 - p_{on})) < 0$.

Рассмотрим второе слагаемое. В силу неравенства (3.3) знаменатель второго слагаемого всегда положителен. Рассмотрим числитель, предварительно преобразовав его:

$$ce^{-\theta c}(p_{on}e^{\theta(2c-r)} + 1 - p_{on} - \frac{r}{c}p_{on}e^{\theta(2c-r)}). \quad (3.4)$$

Для того чтобы b_A монотонно убывала к значению σ_A необходимо, чтобы выражение (3.4) было неотрицательно. Для этого получаем неравенство $p_{on}e^{\theta(2c-r)}(1 - \frac{r}{c}) + 1 - p_{on} \geq 0$. Это верно при $r < c$, так как $p_{on} \in [0,1]$.

Из предыдущих рассуждений, исследование данного вида трафика подразделяется на два случая в зависимости от значений r и c. Рассмотрим подробнее эти случаи.

I. $r < c$

В данном случае b_A монотонно убывает к σ_A увеличением θ. Так как при таком соотношении на ρ_A не наложены никакие условия, то увеличивая значение θ, получаем что $\rho_A \to r$. Тем самым получаем, что оптимизация по θ достигается при значении $\theta = +\infty$ при котором все стохастические параметры стремятся к параметрам в детерминированном случае. Получаем, что при таких ограничениях полученные оценки совпадают.

Ниже представлена таблица проведенных измерений оценок отставания для стохастического и детерминированного сетевого исчисления с параметром $\theta = 10^4$:

Детерминированное	*Стохастическое*	$c, \frac{\text{Мбит}}{\text{с}}$	σ_S, мс	$r, \frac{\text{Мбит}}{\text{с}}$	σ_A, мс	p_{on}	ε'
3.6	*3.6002274*	*100*	*4*	*90*	*0*	*0.5*	*0.1*
3.6	*3.6002968*	*100*	*4*	*90*	*0*	*0.5*	*0.05*
3.6	*3.6002987*	*100*	*4*	*90*	*0*	*0.8*	*0.05*
1.8	*1.8002991*	*100*	*2*	*90*	*0*	*0.8*	*0.05*
1.0	*1.0002991*	*100*	*2*	*50*	*0*	*0.8*	*0.05*

Таблица 1: Измерения оценок отставания в двух сетевых исчислениях.

Результаты стохастического сетевого исчисления хуже детерминированного сетевого исчисления из-за погрешности в тетте, а при оптимальном значении для случая $r < c$ доказано, что оценки совпадают.Результаты стохастического сетевого исчисления могут быть лучше детерминированного сетевого исчисления, но не в рассматриваемом примере.

II. $r \geq c$

Если выполнено это условие, то выражение (3.2) меняет знак в зависимости от θ. Следовательно минимум достигается внутри интервала $[0, +\infty)$. То есть минимизируя по θ функции для отставания и для задержки можем получить оптимальные оценки.

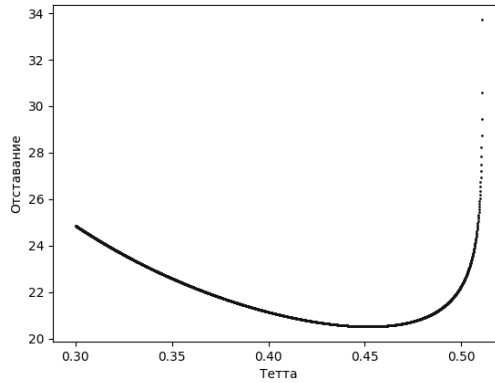

Рис. 4: Зависимость отставания от θ для случая $r \geq c$ в стохастическом сетевом исчислении.

На графике ниже изображена зависимость отставания от θ с входными значениями: $c = 100$ Мбит/с, $r = 101$ Мбит/с, $p_{on} = 0.6$, $\sigma_A = 0$ мс, $\sigma_S = 4$ мс, $\varepsilon' = 0.01$. Вычисления показывают, что минимум достигается при значении $\theta = 0.45$.

Рассмотрим детерминированный случай при выполнении такого условия. Так как $\rho_A = r$, то не выполнено необходимое условие применимости сетевого исчисления $\rho_A \leq \rho_S$. Следовательно, для данного случая мы не можем посчитать оценки для задержки и отставания в детерминированном сетевом исчислении, но можем применить стохастическое. Попробуем получить такие оценки.

Проведем поиск наилучшей оценки, используя программный прототип на языке Python3. Для оптимизации используется библиотека scipy.optimize, а в частности функция minimize, в которой для поиска минимума был выбран метод квадратов. Функции отставания и задержки оптимизируются по одному параметру θ из неограниченной области определения $bnds = (0, +\infty)$, с начальным значением $\delta = 0.001$, где тип необходимого условия – неравенство (ineq) : $\rho_S - \rho_A \geq 0$. Полученное значение θ всегда будет лежать в интервале $(0, 1)$. Но не для любого значения ρ_A таким образом можем вычислить оценку для отставания и задержки, так как максимальная возможная точность вычислений – шаг продвижения по интервалу 10^{-3}. Для определенных значений r, ограничение $\rho_S - \rho_A \geq 0$ будет выполняться для параметра θ на интервале $(0, \delta)$, где δ меньше, чем выбранный шаг продвижения по интервалу в методе оптимизации. Поэтому возникает еще проблема выбора шага продвижения по интервалу, которая не рассматривается в настоящей работе. Тем

самым для значений $c = 100$ Мбит/с, $r = 170$ Мбит/с, $p_{on} = 0.6$, $\sigma_A = 0$ мс, $\sigma_S = 4$ мс, $\varepsilon' = 0.01$ данный алгоритм не может вычислить точную оптимальную оценку для задержки или отставания, так как значения вычисляются в конкретных точках с некоторым фиксированным шагом. Ниже представлена таблица проведенных измерений отставания для стохастического сетевого исчисления (аналогично для задержки):

Оценка отставания	θ оптимальное	$c, \frac{\text{Мбит}}{\text{с}}$	σ_S, мс	$r, \frac{\text{Мбит}}{\text{с}}$	σ_A, мс	p_{on}	ε'
11.8119763	*0.572*	*100*	*4*	*101*	*0*	*0.5*	*0.1*
13.0068470	*0.587*	*100*	*4*	*101*	*0*	*0.5*	*0.05*
37.7351781	*0.194*	*100*	*4*	*101*	*0*	*0.8*	*0.05*
35.7381826	*0.195*	*100*	*2*	*101*	*0*	*0.8*	*0.05*
2815.92099	*0.00349*	*100*	*2*	*150*	*0*	*0.8*	*0.05*

Таблица 3: Измерение отставания в стохастическом сетевом исчислении для случая $r \geq c$.

Тем самым видим, что с увеличением r оценки отставания и задержки растут сверх линейно, в то время как другие параметры влияют незначительно.

4. Заключение

В ходе выполнения настоящей работы был реализован программный прототип анализатора состояния компьютерной сети, который вычисляет оценки задержки и отставания для детерминированного и стохастического сетевого исчисления и проведено экспериментальные сравнения результатов двух исчислений.

Дальнейший интерес для исследования представляют: модификация алгоритма для применения его к более сложной топологии сети, работа с несколькими потоками трафика, проходящими через один обработчик, то есть использование мультиплексирования и планирования трафика и применение данного метода к другим стохастическим моделям представления трафика для моделирования аудио, видео трафика, трафика онлайн-игр и других.

Литература

1. Le Boudec J. Y., Thiran P. *Network calculus: a theory of deterministic queuing systems for the internet.* – Springer Science

and Business Media, 2001. – T. 2050.

2. Jiang Y., Liu Y. *Stochastic network calculus.* – London : Springer, 2008. – T. 1.

3. Чемерицкий Е. В. *Исследование методов контроля функционирования программно-конфигурируемых сетей: Дис. канд. физ-мат. наук: 05.13.11 / Московский Государственный Университет имени М.В. Ломоносова* - 2015 - 200 с.

4. Fidler M. *An end-to-end probabilistic network calculus with moment generating functions.* Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on. – IEEE, 2006. – С. 261-270.

5. Fidler M., Rizk A. *A guide to the stochastic network calculus.* IEEE Communications Surveys & Tutorials. – 2015. – Т. 17. – №. 1. – С. 92-105.

6. Lin C. et al. *Performance evaluation for SDN deployment: an approach based on stochastic network calculus.* China Communications. – 2016. – Т. 13. – №. Supplement. – С. 98-106.

Токарева Е.А., Балашов В.В.

Совместное построение маршрутов и расписаний передачи сообщений в сетях Ethernet с временной синхронизацией[1]

Введение

В данной работе рассматриваются информационно-управляющие системы реального времени, в которых взаимодействие вычислительных задач осуществляется с использованием сети Ethernet с временной синхронизацией [1]. Такие сети примечательны наличием класса сообщений с отправкой по таймеру. Для обмена данными в таких системах необходимо наличие информации о маршрутах передачи данных, а также заранее составленного расписания получения и отправки сообщений на узлах сети.

Существующие методы [2] построения расписания получают на вход предварительно построенные маршруты передачи данных. Маршруты играют ключевую роль при построении расписания, так как при наличии большого количества сообщений, передающихся по одному и тому же физическому каналу, возникают случаи, в которых невозможно построение корректного расписания. При этом зачастую при альтернативном выборе маршрутов построение корректного расписания возможно. В данной статье предложен алгоритм совместного построения маршрутов и расписания передачи данных, то есть маршруты выбираются одновременно с построением расписания и корректируются, если в какой-то момент для выбранных маршрутов построение корректного расписания неуспешно.

Структура данной статьи следующая. Раздел 1 содержит формальную постановку задачи построения маршрутов и расписания передачи данных в сетях Ethernet с временной синхронизацией. В разделе 2 представлен обзор существующих методов, решающих поставленную задачу. Раздел 3 содержит методы решения поставленной задачи, раздел 4 - описание предлагаемого алгоритма совместного построения маршрутов и расписания передачи данных. В разделе 5 приведено экспериментальное исследование алгоритма.

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

1. Схема функционирования сетей с временной синхронизацией

Сети с временной синхронизацией [3] расширяют классический Ethernet. Такие сети состоят из узлов сети, соединенных физическими каналами. Узлы сети представлены набором абонентов и набором коммутаторов. Абонент состоит из коммуникационного контроллера и хоста, на котором выполняются прикладные задачи. Задачи обмениваются сообщениями.

Существует три класса сообщений:

1. TT. Такие сообщения пересылаются сетью в предопределенное время и более приоритетны, чем все остальные типы сообщений. Для каждого такого сообщения предопределены моменты времени его отправки с каждого узла сети его маршрута. Следовательно, для него предопределена и сквозная задержка. При планировании передачи таких сообщений обеспечивается отсутствие очередей на коммутаторах, что позволяет минимизировать задержку при передаче сообщений. Для реализации такого типа сообщений требуются синхронизированные локальные часы на всех узлах сети.

2. RC. Передача сообщений организована по схеме, аналогичной схеме передачи, используемой в протоколе AFDX [4], с учетом наличия в сети более приоритетных TT-сообщений.

3. BE. Сообщения, пересылаемые в соответствии со стандартной политикой Ethernet, имеют наименьший детерминизм задержек при передаче через сеть.

Отправка TT-сообщений происходит согласно заранее построенному расписанию, гарантирующему отсутствие конфликтов, по синхронизированным таймерам. RT-сообщения отправляются в свободное от отправки TT-сообщений время и не влияют на TT-сообщения. BE-сообщения используют оставшуюся пропускную способность.

В случае если на порте коммутатора уже происходит отправка сообщения RC или BE на момент наступления времени отправки с него TT-сообщения, текущая отправка отменяется, производится отправка TT-сообщения, после чего повторяется отмененная отправка. Таким образом, максимальное время обработки TT-сообщения на коммутаторе заранее известно и учитывается на этапе построения расписания.

В данной работе рассматривается только передача сообщений класса TT.

Для выполнения информационного обмена с использованием TT-сообщений необходимо построить локальное расписание отправки TT-сообщений для каждого узла сети.

2. Формальная постановка задача построения маршрутов и расписания передачи данных в сетях с временной синхронизацией

Входными данными для задачи построения маршрутов и расписания передачи данных в системах реального времени, использующих для передачи данных сети Ethernet с временной синхронизацией, являются структура сети и поток данных в этой сети.

Структура сети задается ориентированным графом G = (V, E), где

- V = A ∪ K - множество узлов сети, каждый узел определяется множеством портов V.P = { P_1, P_2,... , $P_{N.P_A}$ };
- A - множество абонентов: A ={ A_1, A_2,... , A_{N_A} }, |A| = N_A;
- K - множество коммутаторов: K = {K_1,K_2,... ,K_{N_K}}, |K| = N_K;
- E - множество дуг, соединяющих элементы из множества V: E = {E_1,E_2,... ,E_{N_E}}, |E| = N_E. При этом для каждой направленной дуги E_i = (V_a,V_b) существует парная дуга E_j = (V_b,V_a). Такие направленные дуги называются каналами.

Поток данных в сети задается множеством однокадровых сообщений Msg = {Msg_1, Msg_2,... , $Msg_{N_{Msg}}$}, |Msg| = N_{Msg}. Каждое сообщение задается четверкой параметров Msg = {T_{msg}, Src_{msg}, Dst_{msg}, t_{msg}}, где

- T_{msg} – период отправки сообщения;
- $Src_{msg} \in$ A – абонент отправитель;
- $Dst_{msg} \in$ A – абонент получатель;
- t_{msg} – максимальная допустимая сквозная длительность передачи сообщения точка-точка.

Обозначим через Msg_i^n экземпляр сообщения Msg_i, передающийся в интервал времени [n * T_{msg_i}, (n+1) * T_{msg_i}).

Так как прикладные задачи отправляют данные периодически, то достаточно построить расписание только на конечном интервале времени. Далее вновь используется готовое расписание. Такой интервал называется интервалом планирования, его длительность обозначим за PI. Длительность интервала планирования равна наименьшему общему кратному периодов всех сообщений:

$$PI = \text{НОК } \{T_i\}$$

Следовательно, достаточно построить локальные расписания только на данном интервале времени.

Для упрощения задачи, интервал планирования делится на равные слоты. Слот – это интервал времени, в который может осуществляться отправка сообщения. В каждый слот с каждого порта каждого узла сети может быть отправлено не более одного сообщения. Моменты времени начала и окончания слота задаются через глобальное время и совпадают на каждом узле сети.

Длительность слота выбирается таким образом, чтобы время передачи сообщения с одного узла сети на другой не превышало время длительности слота. В случае необходимости передать большой объем данных, данные передаются несколькими сообщениями. Сквозная длительность передачи точка-точка в таком случае определяется как время с момента отправки первого сообщения с абонента-отправителя до приема последнего сообщения абонентом-получателем.

Необходимо построить маршруты передачи для каждого сообщения Msg_i, то есть найти упорядоченный набор

$$Route^i = \{ R^i_1, R^i_2, \dots, R^i_{N^i_R} \}, |Route^i| = N^i_R, \text{ где } R^i_1 = Src_{msg}, R^i_{N^i_R} = Dst_{msg}, \forall \text{ k} \neq \text{m } R^i_m \neq R^i_k, (R^i_k, R^i_{k+1}) \in \text{E}$$

и задать слот отправки каждого экземпляра сообщения Msg^n_i с каждого узла маршрута, то для каждой пары Msg^n_i, V_a определить значение

$\text{Time}(Msg^n_i, V_a) \rightarrow \text{k} \in \text{N}$, где k – номер слота отправки, если $\text{V} \in Route^i$, иначе -1. Данное значение задает номер слота отправки экземпляра сообщения Msg^n_i с узла сети V_a.

Решение задачи корректно, если для любых экземпляров сообщений выполняются следующие ограничения:

- Ни на одном порту ни одного узла сети никакой паре различных сообщений не назначен один и тот же слот отправки:

$$\forall \text{ a, n} \neq \text{m, i} \neq \text{j } \text{Time}(Msg^n_i, V_a.p) \neq \text{Time}(Msg^m_j, V_a.p).$$

- Сообщение может передаваться по каналу в сети только после того, как оно было передано по предыдущему каналу на его пути:

$$\forall R^i_m, R^i_n, \forall \text{ m} < \text{n} \rightarrow \text{Time}(Msg^k_i, R^i_m) < \text{Time}(Msg^k_i, R^i_n)$$

- С каждого следующего узла маршрута отправка происходит строго в следующий слот:

$$\forall \text{ n, i, a } \text{Time}(Msg_i^n, V_{a+1}) = \text{Time}(Msg_i^n, V_a) + 1$$

  Данное ограничение не является обязательным для сетей с временной синхронизацией. Рассмотрение задачи без этого ограничения является предметом дальнейшего исследования.

- Сообщение должно быть доставлено абоненту-получателю за один период:

$$\text{Time}(Msg_i^k, Dst_{msg}) - \text{Time}(Msg_i^k, Src_{msg}) < T_{msg}$$

3. Методы планирования информационного обмена в сетях с временной синхронизацией

Сформулированную выше задачу можно решить разными методами. Эти методы делятся на совместные и последовательные.

Совместные методы строят маршруты и расписание одновременно. Методов совместного решения задачи построения маршрутов и расписания передачи данных обнаружено не было.

При последовательном решении задачи сначала строятся маршруты передачи данных, затем расписание передачи данных. Задача построения расписания для сетей с временной синхронизацией – это NP-полная задача выполнимости [3]. Существует множество методов построения корректного расписания:

- использование SMT-решателей (satisfiability modulo theories, SMT - задача выполнимости формул) [2] [3];
- генетические алгоритмы [5];
- целочисленное линейное программирование[6] [7];
- программирование в ограничениях [8].

Однако все эти методы используют уже готовые маршруты потоков данных. В случае, когда маршруты строятся отдельно от расписания, возможна ситуация, когда на один физический канал назначено много потоков данных, не сочетаемых по периодам. Это приводит к невозможности построить корректное расписание.

Таким образом, актуально исследование метода совместного построения маршрутов передачи сообщений и расписания их выдачи, с учетом влияния маршрутов на построение расписания.

4. Алгоритм совместного построения маршрутов передачи данных и расписания обменов

Алгоритм основан на жадных стратегиях [9] и стратегиях ограниченного перебора.

Жадная стратегия обработки сообщений выбрана по аналогии со схемой планирования вычислений Rate Monotonic, в которой в первую очередь планируются задачи с минимальным периодом.

В алгоритме используется понятие **априорной загрузки** однонаправленного канала. Она необходима для определения потенциальных узких мест в сети на этапе, когда неизвестны маршруты передачи данных. Априорная загрузка канала вычисляется как потенциальный объем проходящих через него данных. Для этого рассматривается суммарный объем данных, передающийся за интервал планирования, всех потоков данных, которые потенциально могут проходить через него. При этом учитывается каждый маршрут, в котором содержится этот канал. Предположим, что число существующих маршрутов n. Далее для каждой пары абонентов суммируется весь объем пересылаемых данных за интервал планирования – W. Считается, что по каждому маршруту проходит объем данных W/n. После чего на каждом канале маршрута увеличиваем априорную загрузку на W/n. Таким образом, определяется априорная загрузка для каждого канала сети, позволяя вычислять потенциально перегруженные каналы.

Длина маршрута L определяется как максимальная априорная нагрузка среди всех каналов, входящих в маршрут.

Вес канала определяется как сумма частот уже размещенных в ходе работы алгоритма сообщений, передающихся по данному каналу.

4.1 Общая схема алгоритма

Шаг 1. Упорядочить сообщения Msg_i по по возрастанию периода отправления сообщений T_i. Алгоритм осуществляет последовательную обработку сообщений, поэтому начинать необходимо с наиболее

часто передающихся сообщений, так как редко передающиеся сообщения легче добавить в расписание.

Шаг 2. Для каждого сообщения строится набор допустимых маршрутов $Route^i$ методом k-кратчайших маршрутов по метрике количество каналов в маршруте [10], причем количество маршрутов не должно превышать заранее выбранное число $MaxRoute$. Значение параметра $MaxRoute$ выбирается исходя из сложности алгоритма.

Шаг 3. Для маршрутов вычисляется их длина L($Route^i$), после чего они упорядочиваются по возрастанию длины.

Шаг 4. Для каждого сообщения выполняются шаги 4.1, 4.2.

Шаг 4.1. Для каждого маршрута из набора допустимых для данного сообщения осуществляется построение расписания, пока не будет найден маршрут, для которого оно завершится успешно. Построение расписания для маршрутов осуществляется по следующей схеме: на маршруте выбирается канал $E_i = (V_a,V_b)$ с наибольшим весом, для него запускается построение локального расписания. Эту операцию осуществляет процедура «Построение локального расписания».

Шаг 4.2. Если ни для одного маршрута построение локального расписания не завершилось успехом, СТОП: построить расписание не удалось.

Шаг 5. Если для каждого сообщения был найден маршрут, на котором успешно было построено локальное расписание, то СТОП: успешное завершение алгоритма.

4.2 Построение локального расписания

Шаг Л1. Формируется множество номеров допустимых первых слотов First = {x $\in \overline{0, T_i - 1}$,x-свободный}. Слот называется свободным, если нет никакого экземпляра никакого сообщения Msg_k^m, для которого назначена в этот слот отправка с этого порта данного узла сети. Множество упорядочено по возрастанию номера слота.

Шаг Л2. Для каждого слота из множества допустимых слотов проверяются слоты (First[j] + m * T_i), где m - целое неотрицательное число. Если все такие слоты свободны, то локальное расписание считается построенным и функция задается как Time(Msg_i^m,V_a) = First[j] + m * T_i. В случае успешного построения расписания на данном канале рассматриваются каналы справа и слева по маршруту.

Шаг Л3. Для этих каналов запускается построение локального расписания, где в качестве начального слота выбирается слот First[j] – 1, если рассматривается канал, предшествующий рассматриваемому, и First[j] + 1 иначе. В случае неуспешного построения алгоритм переходит на Шаг 5 и строится расписание для следующего допустимого маршрута.

Примечание: При выполнении шага 4.1. для очередного сообщения построение расписания производится с учетом загрузки каналов расписаниями, построенными для ранее обработанных маршрутов.

Сложность данного алгоритма

$$O(N_{Msg} * log(N_{Msg})) + O(MaxRoute * log(MaxRoute)) + MaxRoute * T_{max} * O(MaxLength * PI),$$

где

- $N_{Msg} = |Msg|$ - количество сообщений;
- MaxLength - максимальная длина маршрута;
- T_{max} - максимальный период сообщения;
- PI - длина интервала планирования.

5. Экспериментальное исследование построенного алгоритма

5.1 Цель исследования

Экспериментальное исследование проводилось с целью сравнить результаты построения расписания совместно с построением маршрутов и последовательно.

Для проведения исследования было выбрано несколько конфигураций системы. Каждая конфигурация содержит информацию о структуре сети и о передающихся сообщениях.

Для каждой конфигурации строятся маршруты и расписание передачи данных двумя методами. В одном случае используется разработанное средство. Во втором - сначала строятся маршруты передачи данных, а затем расписание обменов.

Для построения маршрутов во втором случае используется САПР AFDX [11]. Эта система автоматизированного проектирования позволяет строить маршруты передачи сообщений по сети, накладывая единственное ограничение при построении, которое звучит следующим образом: «Суммарная требуемая пропускная способность на физических каналах передачи данных не должна превышать максимальную доступную пропускную способность». Так как для исследования не требуется накладывать дополнительные ограничения на маршруты, данное средство подходит для решаемой задачи.

Расписание для построенных маршрутов строится с помощью SMT-решателя, так как это наиболее широко используемый метод построения расписания для уже построенных маршрутов. Также к достоинствам этого метода можно отнести детерминированность, которой отдается предпочтение при проектировании бортовых систем. В данной работе был использован SMT-решатель JavaSMT [12].

Два указанных выше подхода (совместное и раздельное построение маршрутов и расписания) сравниваются по критерию успешности построения расписания.

5.2 Исходные данные

Наборы исходных данных состоят из описания структуры сети и описания рабочей нагрузки. Рабочая нагрузка включает в себя описание структуры сети и описание набора сообщений.

Так как бортовые сети являются сетями небольшого размера, для экспериментальных исследований были выбраны структуры сети, представленные ниже на рисунках 1-2. Кругами обозначены коммутаторы, в прямоугольниках – абоненты.

В конфигурациях используется небольшое количество сообщений, однако получатели и отправители были выбраны таким образом, чтобы возникало как можно больше маршрутов с потенциальными конфликтами, то есть маршрутов, при выборе которых построение расписания завершится с ошибкой.

В первой конфигурации используется с сети, представленная на рис.1. Данные о сообщениях представлены в таблице 1. $MaxRoute$ равен 4.

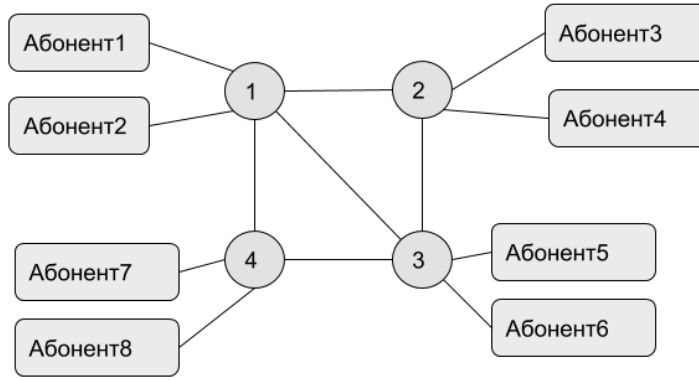

Рис. 1. Структура сети №1

Отправитель	Получатель	Период
A3	A1	2
A4	A2	3
A5	A2	3
A6	A2	3

Таблица 1: Данные о сообщениях 1.

Во второй конфигурации используется структура сети, представленная на рис.1. Данные о сообщениях представлены в таблице 2. $MaxRoute$ равен 4.

Отправитель	Получатель	Период
A3	A1	2
A4	A1	4
A5	A2	3
A6	A2	3

Таблица 2: Данные о сообщениях 2.

В третьей конфигурации используется структура сети, представленная на рис.2. Данные о сообщениях представлены в таблице 3. $MaxRoute$ равен 4.

Отправитель	Получатель	Период
A1	A3	2
A2	A6	3
A5	A4	3
A7	A4	4
A8	A4	4

Таблица 3: Данные о сообщениях 3.

В четвертой конфигурации используется структура сети, представленная на рис.2. Зависимость по данным задана в таблице 4. $MaxRoute$ равен 4.

Отправитель	Получатель	Период
A3	A7	3
A1	A7	3
A5	A7	3
A2	A8	4
A4	A8	4
A6	A8	4

Таблица 4: Данные о сообщениях 4.

Пятая конфигурация по структуре сети и зависимостям по данным идентична конфигурации 4 за исключением выбора параметра $MaxRoute$. Здесь этот параметр принимал значение 1.

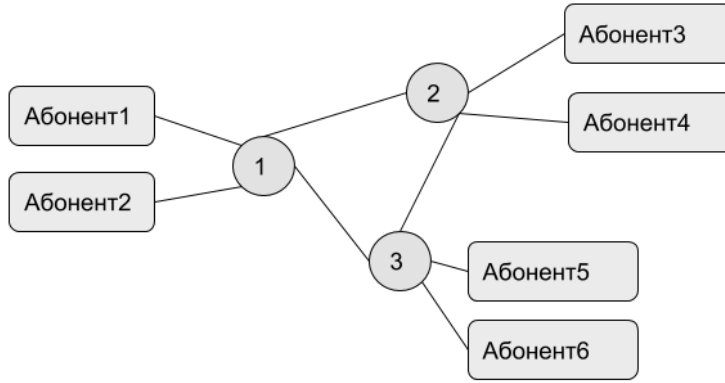

Рис. 2. Структура сети №2

5.3 Результаты экспериментов

На таблице 5 представлены результаты работы последовательного построения и алгоритма для каждой конфигурации системы.

5.4 Выводы

Экспериментальное исследование показало, что использование маршрутов, построенных совместно с расписанием обменов, позволяет в большем количестве конфигураций успешно построить расписание. Однако важно правильно задавать параметр $MaxRoute$, так

Номер конфигурации	Успех совместного построения	Успех последовательного построения
1	+	-
2	+	+
3	+	-
4	+	+
5	-	+

Таблица 5: Результаты экспериментального исследования.

как чем меньше значение этого параметра, тем меньше рассматривается маршрутов, тем меньше вероятность построить корректное расписание, однако большое значение этого параметра увеличивает сложность алгоритма.

В данной серии экспериментов можно сделать вывод, что отсутствие перебора маршрутов ($MaxRoute = 1$) снижает возможности алгоритма по построению расписаний.

Подход к выбору значения $MaxRoute$ в зависимости от характеристик входных данных может быть предметом дальнейшего исследования.

6. Заключение

Сети Ethernet с временной синхронизацией позволяют обеспечить высокий временной детерминизм обмена данными в информационно-управляющих системах реального времени. При планировании обмена по таким сетям необходимо построить маршруты передачи сообщений и сформировать расписания выдачи сообщений для всех узлов сети, включая абонентов и коммутаторы. В существующих методах планирования обмена для сетей рассматриваемого класса вначале строятся маршруты передачи сообщений, затем для полученного набора маршрутов строятся расписания выдачи сообщений.

Подобное разделение этапов приводит в ряде случаев к невозможности построения расписаний в связи с перегрузкой отдельных участков сети.

Предложенный в данной работе подход совмещает построение маршрутов и расписаний выдачи сообщений, что позволяет учиты-

вать загрузку сетевых каналов при построении маршрутов. Экспериментальное исследование показало преимущество предложенного подхода над раздельным выполнением этапов планирования. К направлениям дальнейшего исследования можно отнести модификацию разработанного алгоритма с тем, чтобы при неуспешном построении решения выполнялось достраивание маршрутов и расписаний с частичным нарушением ограничений; для построенных таким образом маршрутов будет применяться известный метод построения расписания при помощи SMT-решателя.

Литература

1. Obermaisser R. *Time-triggered communication.* – CRC Press, 2011.

2. Steiner W. *An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks.* // Real-Time Systems Symposium (RTSS), 2010 IEEE 31st. – IEEE, 2010. – P. 375-384.

3. Steiner W. *TTEthernet dataflow concept* // Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on. – IEEE, 2009. – P. 319-322.

4. AFDX / ARINC 664 Tutorial (1500-049) // Condor Engineering, Inc. – 2005.

5. Monnier Y., Beauvais J.P., Deplanche A.M. *A genetic algorithm for scheduling tasks in a real-time distributed system* // Euromicro Conference, 1998. Proceedings. 24th. – IEEE, 1998. – vol. 2. – P. 708-714.

6. Jones J.G., Meneghin B.J., Kirby M. W. *Formulating adjacency constraints in linear optimization models for scheduling projects in tactical planning* // Forest science. – 1991. – vol. 37. – No. 5. – P. 1283-1297.

7. Goswami D. *Time-triggered implementations of mixed-criticality automotive software* – EDA Consortium, 2012. – P. 1227-1232.

8. Marriott K., Stuckey P.J. *Programming with constraints: an introduction.* – MIT press, 1998.

9. Вдовин П.М. *Жадные алгоритмы и стратегии ограниченного перебора для планирования вычислений в системах с жесткими требованиями к качеству обслуживания* : дис. ... канд. физ.-мат. наук: 05.13.11. – М., 2016. – 127 с.

10. Yen J. Y. Finding the k shortest loopless paths in a network // Management Science. – 1971. – vol. 17. – № 11. – P. 712-716.

11. Вдовин П.М. *Инструментальная система проектирования сетей AFDX* // труды МФТИ. 2015. Т. 7. No 2. С.131 – 136.

12. JavaSMT [HTML] https://github.com/sosy-lab/java-smt

Чернышева А.Ю., Швецов Д.А., Шалимов А.В.

Расширение системы Maple для работы с несколькими OpenFlow таблицами

1. Введение

1.1 Программно-конфигурируемые сети

Программно-конфигурируемы сети – новый вид архитектуры сетей, позволяющий облегчить управление сетевыми устройствами за счет наличия контроллера – устройства централизованного управления [1, 5, 9]. Наиболее популярной реализацией ПКС является ПКС с использованием протокола OpenFlow [2, 10]. Это довольно низкоуровневый протокол, что делает его недостаточно удобным в использовании и может приводить к большому числу ошибок. Чтобы этого не происходило, проводится разработка более высокоуровневых абстракций и языков. Одной из таких реализаций является система Maple [3], которая используется в открытой версии контроллера Runos [11]. Тем не менее, в тех случая, когда Maple работает, устанавливая правила только в одну таблицу каждого коммутатора, это приводит к появлению избыточных правил. В ходе данной статьи был разработан и применен к контроллеру Runos алгоритм работы Maple с несколькими OpenFlow таблицами. Также были проведены экспериментальные исследования, подтвердившие сокращение общего числа правил в три раза.

Для облегчения управление сетью и повышения ее производительности и функциональности был придуман новый вид архитектуры сетей, а именно – Программно-Конфигурируемые сети. Основная идея данного подхода заключается в отделении уровня управления сетью от устройств передачи данных и перенесении его на единое сетевое устройство, названное контроллером [1].

При этом необходимо иметь новый унифицированный протокол, связывающий эти два уровня.

1.2 OpenFlow

OpenFlow не единственный, но самый популярный протокол, поддерживающий связь коммутаторов с контроллером. Использование OpenFlow подразумевает [2]:

- Наличие в сети безопасного канал, ведущего от каждого коммутатора к контроллеру, для обмена служебными сообщениями;
- Наличие таблиц OpenFlow правил на каждом коммутаторе, задаваемые контроллером;

1.3 Недостатки программирования приложений с использованием OpenFlow

Так как OpenFlow разрабатывался как протокол взаимодействия уровня управления с уровнем передачи данных, написание приложения напрямую через инструменты OpenFlow сопряжено с трудностями для разработчиков логики сети.

Программисту нужно постоянно отслеживать зависимости, запоминать большое число низкоуровневых команд, типов, следить за отсутствием перекрытий OpenFlow правил и за правильной расстановкой их приоритетов, из-за чего код становится плохо структурируемым и не таким понятным, как его более высокоуровневые аналоги. Все это может приводить к большому числу чисто технических ошибок и отвлекать от основной идеи алгоритма.

Для того, чтобы решить эту проблему, были разработаны более высокоуровневые системы, скрывающие низкоуровневые детали и позволяющие облегчить задачу написания приложений разработчикам. К таким системам относятся Pyretic [7], Frenetic [6], Maple [3], Procera [8], FML [4]. В рамках данной работы будет рассматриваться только одна система - Maple.

1.4 Недостаток работы системы Maple с одной таблицей

В ходе своей работы, система Maple генерирует таблицу потоков по высокоуровневым командам контроллера. В результате, мы получаем одну большую таблицу с множеством правил, которая может не поместиться в память коммутатора целиком и будет неудобна в использовании. Так как протокол OpenFlow поддерживает использование нескольких таблиц, проанализируем на примере, что будет происходить, если производить запись правил в две таблицы вместо одной.

Пусть мы рассматриваем два поля исходного пакета, каждое из которых принимает 3 значения. Если мы будем записывать правила в одну общую таблицу, то получим $3^2 = 9$ записей. Если же добавить в первую таблицу проверку значений первого поля пакета, во вторую таблицу - проверку значений второго поля пакета и по очереди обходить таблицы, то получим $3 * 2 = 6$ записей.

Таким образом, мы получаем существенный выигрыш даже для такого небольшого диапазона значений полей.

2. Постановка задачи

В ходе данной работы требовалось:

1. Разработать алгоритм для работы Maple с несколькими OpenFlow таблицами;
2. Реализовать алгоритм в открытой версии контроллера Runos и провести экспериментальные исследования;

3. Описание работы системы Maple

3.1 Система Maple

Основная идея Maple заключается в том, что мы вводим абстракцию следующего содержания: будем считать, что каждый пакет, попадающий в сеть, проходит через некоторую функцию обработчик (в нашем случае в качестве одной такой функции будет выступать приложение контроллера). При написании обработчика мы можем использовать следующие методы [3]:

- `load` - считывает поле пакета;
- `test` - проверяет поле пакета на соответствие заданному константному значению;

По завершении работы функция возвращает действие, которое нужно совершить над пакетом:

- Отправить на контроллер;
- Отправить на заданный порт;
- Отбросить пакет;

Чтобы избежать прохождения всех пакетов через контроллер, можно выделять некоторое общее подмножество проверяемых полей для группы пакетов и запоминать полученный результат. Для этой цели, Maple использует систему кэширования, реализованную в виде дерева Maple.

3.2 Дерево Maple и его построение

Построение трассы

Для того чтобы построить правило, нужно знать поля, к которым обращалось приложение, и итоговое действие над пакетом. Структура, состоящая из последовательности таких обращений с добавлением результата функции в конец, называется трассой. Чтобы представить собранные трассы как единое целое, используется структура дерева Maple (будем называть ее также $TraceTree$) [3].

Структура $TraceTree$

Будем считать, что пакет имеет поля $a_1 \dots a_n$, и записывать как $p.a$ значение поля a для пакета p. Будем обозначать через $dom(a)$ набор возможных значений поля a: $p.a \in dom(a)$.

Определение $TraceTree$: $TraceTree$ – это корневое дерево, в котором каждой вершине t соответствует поле $type_t$, чье значение может быть одним из следующих: $L(leaf)$, $V(value)$, $T(test)$, $\Omega(empty)$.

- Если $type_t = L$, то вершина содержит поле $action_t$, которое представляет собой действие, совершаемое над пакетом;
- Если $type_t = V$, то вершина содержит поля $attr_t$ и $subtree_t$, где $subtree_t$ – ассоциативный массив, такой что $subtree_t[v]$ указывает на поддерево $TraceTree$, а $v \in keys(subtree_t)$. Если при выполнении программы пакет удовлетворяет условию $p.attr_t = v$, то она продолжает выполняться в $subtree_t[v]$;
- Если $type_t = T$, то t содержит поля $attr_t$, $value_t$, такие что $value_t \in dom(attr_t)$, и два поддерева t_+ и t_-. Если при выполнении программы для пакета p верно условие $p.attr_t = value_t$, то работа продолжается по ветви t_+, иначе по ветви t_-;
- Если $type_t = \Omega$, то t не имеет полей;

Для каждого приложения построение дерева начинается с единственной пустой вершины Ω.

Для каждого нового пакета, проходящего через функцию обработчик, строится трасса, после чего Maple представляет ее как ветвь дерева (в приведенном ниже алгоритме это метод $TraceToTree(trace)$) и добавляет к дереву $TraceTree$. Алгоритм проходит по дереву и трассе одновременно, чтобы найти место для добавления ветви. К найденной вершине он добавляет оставшуюся часть трассы.

В алгоритме (Рис. 1) также используются методы $head(trace)$, чтобы найти первый элемент трассы, и $next(trace)$, чтобы его удалить и вернуть оставшуюся часть трассы.

```
1  Function AugmentTT(t, trace)
2      if type_t = Ω then
3          t ← TraceToTree(trace); return
4      end
5      repeat
6          item = head(trace); trace ← next(trace);
7          if type_t = T then
8              if item.outcome is True then t_± ← t_+ else t_± ← t_−;
9              if type_{t_±} = Ω then
10                 t_± ← TraceToTree(trace); return
11             else
12                 t ← t_±;
13             end
14         end
15         else if type_t = V then
16             if item.value ∈ keys(subtree_t) then
17                 t ← subtree_t[item.value];
18             else
19                 t ← TraceToTree(trace); return
20             end
21         end
22     until;
```

Рис. 1: Добавление ветви к дереву $TraceTree$

Компиляция дерева Maple в таблицу потоков

После построения дерева Maple, необходимо перенести полученные решения в правила OpenFlow.

Компиляция в таблицу представляет собой рекурсивный обход дерева, на каждом узле которого изменяется множество $match$, пред-

ставляющее из себя набор полей и значений, по которым определяется поток (Рис. 2).

```
1  Function BuildFT(t)
2  |  priority ← 0;
3  |  BUILD(t, any);
4  |  return
5  Function BUILD(t, m)
6  |  if type_t = L then
7  |  |  emitRule(priority, m, action_t);
8  |  |  priority ← priority + 1;
9  |  end
10 |  else if type_t = V then
11 |  |  foreach v ∈ keys(subtrees_t) do
12 |  |  |  BUILD(subtrees_t[v], m ∧ (attr_t : v));
13 |  |  end
14 |  end
15 |  else if type_t = T then
16 |  |  BUILD(t_−, m);
17 |  |  m_t = m ∧ (attr_t : value_t);
18 |  |  emitRule(priority, m_t, ToController);
19 |  |  priority ← priority + 1;
20 |  |  BUILD(t_+, m_t);
21 |  end
```

Рис. 2: Построение таблицы потоков по дереву $TraceTree$

4. Алгоритм работы Maple с несколькими таблицами

Для начала, опишем, по какому принципу реализована работа системы Maple на контроллере Runos с одной таблицей. Когда пакет попадает на контроллер, он последовательно проходит через все приложения. В результате строится единая трасса, состоящая из вызовов `load` и `test` каждого приложения, и решения для данного пакета. Далее, согласно алгоритму, дерево дополняется новой ветвью и происходит его компиляция в одну таблицу для каждого из коммутаторов (Рис. 3).

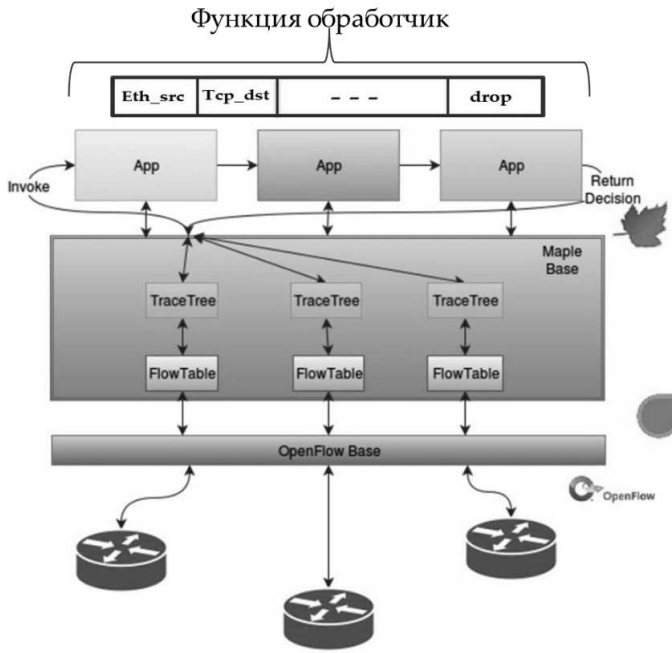

Рис. 3: Схема работы системы Maple с одной таблицей

Чтобы расширить этот алгоритм для работы с несколькими таблицами, будем применять все описанные выше шаги исходного алгоритма, а именно: строить трассу и создавать дерево - для каждого приложения в отдельности. Компиляция при этом проводится в таблицу на коммутаторе с соответствующим порядковым номером, т.е. первое приложение устанавливает правила в первую таблицу, второе – во вторую и т.д (Рис. 4). В качестве действия в конце каждого правила каждой таблицы OpenFlow, кроме последней, устанавливается дополнительная информация о переходе на следующую таблицу. Если в одном из приложений присутствовало указание не учитывать решение последующих за ним приложений, указатель на следующую таблицу установлен не будет.

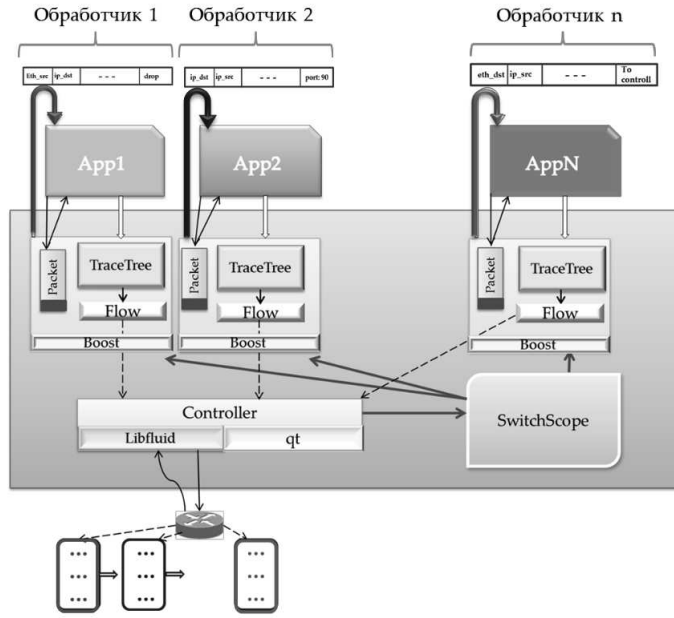

Рис. 4: Общая схема работы Maple с несколькими таблицами

Рассмотрим, как происходит заполнение таблиц более подробно:

1. Изначально в каждой таблице устанавливается правило, в соответствии с которым любой пакет, попадающий на коммутатор, отправляется на контроллер, а каждое соответствующее дерево состоит только из пустой вершины.

2. После прихода первого пакета на коммутатор, он отправляется на контроллер в соответствии с правилом, установленным в первой таблице. Там он проходит через все функции обработчики после чего:

   (a) Если данная функция определила действие для этого пакета, в таблицу устанавливается соответствующее правило (с приоритетом выше, чем у правила, посылающего любой пакет на контроллер). К дереву Maple, по которому была скомпилирована эта таблица, добавляется новая ветвь. К числу действий, производимых над пакетом, добавляется переход на следующую таблицу (это не относится к правилам из последней таблицы).

(b) Если функция обработчик, т.е. приложение, не определила дальнейшего действия над пакетом, то в качестве действия в правило добавляется только переход на следующую таблицу. Если при этом при изучении пакета были вызваны методы `load` и/или `test`, они будут отражены в построении дерева (будут присутствовать вершины типа $Test$ и $Value$, но будет отсутствовать финальная вершина типа $Leaf$).

(c) Если несколько приложений хотят выполнить разные действия для одного пакета, то они выполняются последовательно.

(d) Если в приложении присутствовало указание не учитывать решение следующих за ним обработчиков, переход на следующую таблицу для этого потока не устанавливается.

3. С каждым поступлением нового пакета, действия над ним будут определяться в соответствии с правилами, установившимися в таблицах:

   (a) Если правила для этого пакета уже определены каждым из приложений, отправка на контроллер производиться не будет, а будут выполняться действия из таблиц.

   (b) Если поступит пакет, и по правилу какой-то таблицы будет отправлен на контроллер, процесс построения трасс начнется для всех приложений, начиная с текущего (не будут проверяться предыдущие приложения, так как в предыдущих таблицах правило для этого пакета уже определено). Можно было бы реализовать такую логику: если в таблице есть указание $ToController$, то отправлять пакет для просмотра только текущим приложением, а по возвращении передавать следующей таблице. Однако в таком случае возникнет большое число запросов на контроллер и эффект от кэширования правил будет не так заметен.

   После того как пакет попал на контроллер, каждый обработчик сформирует свою ветвь и добавит ее к своему дереву. В результате, таблицы потоков соответствующих приложений обновятся. Для определения итогового действия над этим пакетом, необходимо возобновить обход измененных таблиц. При этом будут действовать те же правила установки правил перехода на следующую таблицу, что и в 2.

Ограничение на работу данного алгоритма

Ясно, что вышеприведенный алгоритм будет работать некорректно в случае, если число приложений на контроллере будет превосходить количество таблиц хотя бы на одном из коммутаторов. Эту проблему планируется решить в дальнейшем, а пока предполагается, что число таблиц коммутаторов будет не меньше числа приложений.

Также стоит отметить, что приложения могут изменять поля приходящих пакетов, и это не является ограничением.

5. Программная реализация

В ходе программной реализации, изменениям подверглись два файла исходного кода (полный код проекта можно изучить здесь `https://github.com/A13Chernysheva/Runos`): `Runtime.hh` и `Controller.cc`.

1. Так как в предложенном алгоритме работа самой системы Maple не менялась, была выделена структура данных, реализующая Maple - класс *Runtime*. Внутри себя он содержит:

   (a) функцию обработчик (поле *Policy*), по которой строится трасса;

   (b) дерево Maple (*TraceTree*);

   (c) класс *Backend*, методы которого осуществляют установку правила в таблицу с соответствующим номером или удаление правила;

   После некоторых изменений формата поля *Policy*, стало возможным, чтобы класс *Runtime* мог работать с одним единственным приложением.

2. В структуре потока *Flow* (описанного в файле `Controller.cc`) было добавлено поле логического типа. В зависимости от этого значения происходит установка инструкции для потока *FlowMod* о переходе или не переходе к следующей таблице.

3. При подключении коммутатора к контроллеру на контроллере создается его оболочка (структура *SwitchScope*). Изначально, в этой оболочке хранился один экземпляр класса *Runtime*, описанный в 1, через который происходило взаимодействие с системой Maple. В данной реализации число классов *Runtime* было увеличено до числа зарегистрированных приложений. Также увеличилось число классов *Backend*, так как каждый такой класс необходимо передать на вход своему *Runtime*.

4. В итоге, при попадании пакета на контроллер, в оболочке соответствующего коммутатора запускается цикл по всем приложениям, внутри которого обрабатывается данный пакет. А именно, происходит вызов метода *Augment*, соответствующего данному приложению экземпляра класса *Runtime*, который инициирует начало работы системы. В качестве параметров ему передается исходный пакет, поток *Flow* данного приложения и функция, реализующая текущее приложение.

6. Экспериментальное исследование

6.1 Цель исследования

Целью экспериментального исследования было оценить число правил в таблицах коммутатора при использовании расширенного алгоритма работы для нескольких таблиц и сравнить полученный результат с соответствующими показателями предыдущей реализации для одной таблицы.

6.2 Тестовое окружение

Исследование проходило в системе эмуляции компьютерной сети `Mininet` [12], в качестве коммутатора использовался `Open vSwitch` [13]. Для генерации траффика была задействована встроенная функция системы `Mininet pingall`.

Модифицированный контроллер тестировался в топологии single (топология, в которой все хосты напрямую соединены с коммутатором и не соединены между собой), с количеством хостов равным 10, 20, 30 и 40. Функции обработчики задавались следующими приложениями:

- `LinkDiscovery` - исследует соединения сети, необходимо для поддержания информации о топологии;
- `ArpHandler` - реализует Arp протокол;
- `HostManager` - запоминает все конечные устройства сети;
- `LearningSwitch` - отправляет пакет по кратчайшему маршруту;

Вследствие своей функциональности, первые два приложения носят вспомогательный характер, а последние два необходимы именно для проверки эффективности разработанного алгоритма.

6.3 Полученные результаты

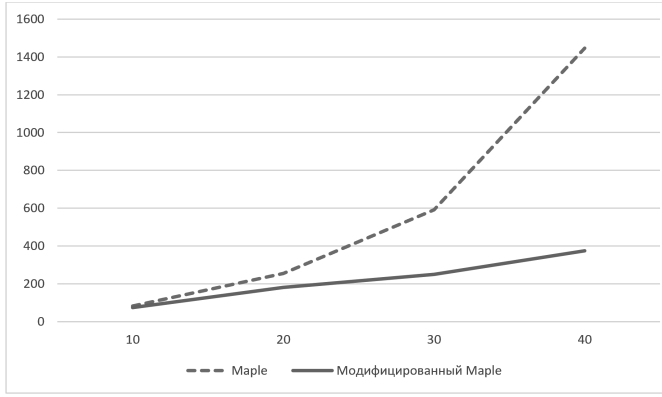

Рис. 5: Зависимость числа правил на коммутаторах от числа устройств в сети

По полученным результатам (Рис. 5) можно сделать вывод, что использование нескольких OpenFlow таблиц сокращает суммарное число правил. При увеличении числа устройств, выигрыш также растет: при проведении экспериментов максимальное значение, которого удалось достичь – 75%.

7. Заключение

В ходе проделанной работы была изучена реализация системы Maple на отечественном контроллере Runos и выявлены ее недостатки. С целью их устранения был разработан и реализован алгоритм работы Maple с несколькими таблицами, а также проведены экспериментальные исследования, показавшие эффективность данного алгоритма.

В дальнейшем планируется избавить данный метод от ограничения на число таблиц в коммутаторах.

Литература

1. Смелянский Р. Л. *Программно-конфигурируемые сети* //Открытые системы. СУБД. – 2012. – Т. 9. – С. 23-26.

2. McKeown N. et al. *OpenFlow: enabling innovation in campus networks* //ACM SIGCOMM Computer Communication Review. – 2008. – T. 38. – No. 2. – C. 69-74.

3. Voellmy A. et al. *Maple: simplifying SDN programming using algorithmic policies* //ACM SIGCOMM Computer Communication Review. – ACM, 2013. – T. 43. – No. 4. – C. 87-98.

4. Shvetsov D., Shalimov A. *Overhead reduction of an automatic rules generation system in an openflow controller*Modern Network Technologies, MoNeTec-2018. — Москва, 2018. — P. 89–96

5. Casado M., Foster N., Guha A. *Abstractions for software-defined networks*//Communications of the ACM. – 2014. – T. 57. – No. 10. – C. 86-95.

6. Foster N. et al. *Frenetic: A network programming language*//ACM Sigplan Notices. – ACM. – 2011. – T. 46. – No. 9. – C. 279-291.

7. Reich J. et al. *Modular sdn programming with pyretic*//Technical Reprot of USENIX. – 2013.

8. Voellmy A., Kim H., Feamster N. *Procera: a language for high-level reactive network control*//Proceedings of the first workshop on Hot topics in software defined networks. – ACM, 2012. – C.43-48.

9. Kreutz D. et al. *Software-defined networking: A comprehensive survey*//Proceedings of the IEEE. – 2015. – T. 103. – No. 1. – C. 14-76.

10. Erickson D. *The beacon openflow controller*//Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. – ACM, 2013. – C. 13-18.

11. Runos `https://github.com/ARCCN/runos`

12. Mininet `https://github.com/mininet/mininet`

13. Open vSwitch `https://www.openvswitch.org`

Аннотации

Pinaeva N.M., Antonenko V.A. Inter-domain Communication Approach in MANO Platform // Программные системы и инструменты. Тематический сборник № 18, М.: Изд-во факультета ВМиК МГУ, 2018.

MANO-platform provides to user service chaining functionality, which consists of multiple network functions. In multi-domain platform different NFs from certain service chain may be deployed in different NFVI domains. It means that MANO-platform should manage inter-domain network which connect different NFVI domains and be able to create service chains over this network. This paper introduces the architecture of multi-domain MANO-platform and the approach to manage and orchestrate inter-domain communication in such platforms.

Ил.: 7 рис., Библиогр.: 16.

Васильева Ю.О., Костенко В.А., Морозов В.В. Планирование вычислений в территориально распределенном ЦОД. // Программные системы и инструменты. Тематический сборник № 18, М.: Изд-во факультета ВМиК МГУ, 2018.

Статья посвящена вопросу организации метапланирования над совокупностью локальных планировщиков ЦОД. Описаны подходы к построению алгоритма метапланирования, указаны различные схемы взаимодействия метапланировщика с локальными планировщиками, приведены результаты исследования одной из схем алгоритма.

Ил.: 1 рис., Библиогр.: 6.

Волканов Д.Ю., Малышев Н.С. Оптимизация надёжности распределённых вычислительных систем с модульной архитектурой. // Программные системы и инструменты. Тематический сборник № 18, М.: Изд-во факультета ВМиК МГУ, 2018.

В статье приведена постановка задачи оптимизации надёжности распределённых вычислительных систем с модульной архитектурой,

приведён обзор алгоритмов решения схожих задач, приведено описание предлагаемого метода и его экспериментальное исследование.

Ил.: 3 табл., Библиогр.: 10.

Ковалёв А.И., Степанов Е.П. Разработка метода адаптивного выбора алгоритмов управления перегрузкой для транспортных соединений // Программные системы и инструменты. Тематический сборник № 18, М.: Изд-во факультета ВМиК МГУ, 2018.

В статье приведена проблема управления перегрузкой в изменчивом сетевом окружении, предложен подход адаптивного выбора алгоритмов управления перегрузкой для решения указанной проблемы, приведено описание разработанной реализации и экспериментальное исследование предложенного подхода.

Ил.: 4 рис., 2 табл., Библиогр.: 12.

Королев В.В., Шалимов А.В. Разработка системы отображения произвольного конвейера OpenFlow таблиц в коммутаторы с единственной таблицей. // Программные системы и инструменты. Тематический сборник № 18, М.: Изд-во факультета ВМиК МГУ, 2018.

В статье приведена постановка задачи разработки и исследования алгоритма трансляции множества OpenFlow таблиц в единственную таблицу в сети, приведён обзор подходов к решению схожих задач, приведено описание предлагаемого метода и его экспериментальное исследование.

Ил.: 1 рис., 5 табл., Библиогр.: 5.

Костенко В.А., Маслов Н.С. Исследование влияния способа выбора начального приближения на точность и вычислительную сложность обучения нейронных сетей. // Программные системы и инструменты. Тематический сборник № 18, М.: Изд-во факультета ВМиК МГУ, 2018.

В данной статье рассматривается проблема влияния способа выбора начального приближения на точность и вычислительную сложность обучения нейронных сетей на примере свёрточных сетей для распознавания изображений. Проведён сравнительный анализ методов инициализации весовых коэффициентов и алгоритмов обучения, выполнено экспериментальное исследование, показавшее целесообразность использования в общем случае методов с адаптивной настройкой параметров для инициализации и обучения нейронных сетей.

Ил.: 6 рис., Библиогр.: 14.

Петров И.С., Шендяпин А.С. Обзор средств обеспечения анонимности в SDN. // Программные системы и инструменты. Тематический сборник № 18, М.: Изд-во факультета ВМиК МГУ, 2018.

В статье приведено описание основных критериев сравнения сервисов анонимности, приведён обзор систем обеспечения анонимности, их достоинств и недостатков, приведено сравнение рассмотренных систем между собой.

Ил.: 1 табл., Библиогр.: 12.

Синякова М.А., Степанов Е.П. Анализатор состояния компьютерной сети на основе теории стохастического сетевого исчисления. // Программные системы и инструменты. Тематический сборник № 18, М.: Изд-во факультета ВМиК МГУ, 2018.

Настоящая работа посвящена изучению теории стохастического сетевого исчисления и сравнению оценок, полученных с использованием стохастического сетевого исчисления, с оценками, вычисленными с использованием детерминированного сетевого исчисления. Было проведено экспериментальное исследование предложенного алгоритма, в котором сравнивались полученные оценки с оценками детерминированного сетевого исчисления.

Ил.: 5 рис., 2 табл., Библиогр.: 6.

Токарева Е.А., Балашов В.В. Совместное построение маршрутов и расписаний передачи сообщений в сетях Ethernet с временной синхронизацией. // Программные системы и инструменты. Тематический сборник № 18, М.: Изд-во факультета ВМиК МГУ, 2018.

В статье приведена постановка задачи построения маршрутов и расписаний обмена данными в сетях с временной синхронизацией, рассмотрены существующие методы построения расписания в таких сетях, приведено описание предлагаемого метода и его экспериментальное исследование.

Ил.: 2 рис., 5 табл., Библиогр.: 12.

Чернышева А.Ю., Швецов Д.А., Шалимов А.В. Расширение системы Maple для работы с несколькими OpenFlow таблицами. // Программные системы и инструменты. Тематический сборник № 18, М.: Изд-во факультета ВМиК МГУ, 2018.

В статье приведено описание работы системы Maple, предлагается алгоритм, позволяющий данной системе работать с несколькими таблицами, описаны основные этапы программной реализации, а также проведенные экспериментальные исследования.

Ил.: 5 рис., Библиогр.: 13.

Software systems and tools : Periodical / Ed. R. L. Smelyansky. – Moscow : Publishing Department of the Faculty of Computational Mathematics and Cybernetics (license ID № 05899 from 24.09.2001); MAKS Press, 2018. – № 18. – 132 p.

ISBN 978-5-89407-594-5 (CMC MSU)

ISBN 978-5-317-06026-8 (MAKS Press)

Theese proceedings consists of student's works, who have received the recommendation of the Department's scientific seminars, which he created and directed Korolev L.. This edition continues the tradition of publishing in memory of this outstanding man. The proceedings contains articles devoted to creating information computing infrastructure for scientific research, problems of modern computer networks, methods and tools for organizing and managing cloud computing, tools that support the operation of real-time systems (RTS), methods for user authentication and the analysis of their behavior in computer networks in the interests of information security.

These papers will be of interest to students, graduate students and professionals in the development of application software systems using new information technologies.

Keywords: information telecommunication technology, software-defined networking, OpenFlow switch, centralized controller, wireless Wi-Fi networks, cloud computing, data center, virtual resources, network functions virtualization, cloud platform, real-time systems, integrated modular avionics, job scheduling, convolutional neural networks, evolutionary algorithms, reliability optimization problem.

Научное издание

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 18

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

Издательство «МАКС Пресс»
Главный редактор: *Е. М. Бугачева*

Напечатано с готового оригинал-макета
Подписано в печать 25.12.2018 г.
Формат 60x90 1/16. Усл.печ.л. 8,25.
Тираж 60 экз. Заказ 332.

Издательство ООО “МАКС Пресс”
Лицензия ИД N 00510 от 01.12.99 г.
119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус, 527 к.
Тел. 8(495)939-3890/91. Тел./Факс 8(495)939-3891.

Отпечатано в полном соответствии с качеством
предоставленных материалов в ООО «Фотоэксперт»
115201, г. Москва, ул. Котляковская, д.3, стр. 13.