ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ТЕЛЕКОММУНИКАЦИИ
СПбГУТ))) Апрель 2020 Том 8 № 1 ISSN 2307-1303 http://www.itt.sut.ru
TELECOM IT
April 2020 Vol. 8 Iss. 1 ISSN 2307-1303 http://www.itt.sut.ru

ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ БАЛАНСИРОВЩИКА НАГРУЗКИ В ОБЛАЧНЫХ МИКРОСЕРВИСХНЫХ АРХИТЕКТУРАХ

В. С. Елагин*, В. Е. Николаев

Санкт-Петербургский государственный университет телекоммуникаций
им. проф. М. А. Бонч-Бруевича Санкт-Петербург, 193232, Российская Федерация
*Адрес для переписки: elagin.vas@gmail.com

Аннотация—Для любой облачной микросервисной архитектуры одной из основополагающих характеристик является масштабируемость. В современно мире, любой сервис, который предназначен для предоставления услуг широкому числу потребителей, должен отличаться хорошей масштабируемостью и быстротой скоростью ответа. Потенциальные пользователи ни за что не захотят терпеть постоянные ошибки, потери запросов, отказ в обслуживании и другие проблемы, связанные с нехваткой производительности сервиса. **Предмет исследования**: система балансировки нагрузки при обработке запросов и взаимодействия в рамках облачной микросервисной архитектуры. **Метод:** внедрение дополнительных элементов в облачную микросервисную архитектуру для обеспечения балансировки нагрузки. **Основные результат:** Проведено сравнение ряда популярных алгоритмов балансировки нагрузки в облачных микросервисхных структурах, а также предложены варианты модернизации в облачных микросервисхных архитектур при балансировке трафика на различных уровнях модели OSI. **Практическая значимость:** в рамках исследования была решена задача увеличения объема обрабатываемых запросов, не используя пропорциональное увеличение аппаратных ресурсов

Ключевые слова—Микросервисная архитектура, виртуализация, балансировка, динамическое масштабирование ресурсов высоконагруженные приложения.

Информация о статье
УДК 004.725.4
Язык статьи – русский.
Поступила в редакцию 20.01.2020, принята к печати 10.04.20.

Ссылка для цитирования: Елагин В. С., Николаев В. Е. Функциональное назначение балансировщика нагрузки в облачных микросервисхных архитектурах // Информационные технологии и телекоммуникации. 2020. Том 8. № 1. С. 67–75. DOI 10.31854/2307-1303-2020-8-1-67-75.

FUNCTIONAL PURPOSE OF THE LOAD BALANCER IN CLOUD MICROSERVICE ARCHITECTURES

V. Elagin*, V. Nikolaev

The Bonch-Bruevich Saint-Petersburg State University of Telecommunications,
St. Petersburg, 193232, Russian Federation
*Corresponding author: elagin.vas@gmail.com

Abstract—Scalability is a fundamental characteristic of any cloud-based microservice architecture. In the modern world, any service that is designed to provide services to a wide number of consumers should be characterized by good scalability and fast response speed. Potential users will never want to suffer constant errors, lost requests, denial of service, and other problems associated with a lack of service performance. **Research subject**: load balancing system in query processing and interaction within the cloud microservice architecture. **Method**: implementing additional elements in the cloud microservice architecture to provide load balancing. **Core results**: a number of popular load balancing algorithms in cloud microservice structures are compared, and options for upgrading in cloud microservice architectures for traffic balancing at different levels of the OSI model are proposed. **Practical relevance**: the study solved the problem of increasing the volume of processed requests without using a proportional increase in hardware resources

Keywords—Micro service architecture, virtualization, balancing, high-load applications, dynamic resource scaling.

Article info
Article in Russian.
Received 20.01.2020, accepted 10.04.20.

For citation: Elagin V., Nikolaev V.: Functional Purpose of the Load Balancer in Cloud Microservice Architectures // Telecom IT. 2020. Vol. 8. Iss. 1. pp. 67–75 (in Russian). DOI 10.31854/2307-1303-2020-8-1-67-75.

Введение

Для любой облачной микросервисной архитектуры одной из основополагающих характеристик является масштабируемость.

В современно мире, любой сервис, который предназначен для предоставления услуг широкому числу потребителей, должен отличаться хорошей масштабируемостью и быстротой скоростью ответа. Потенциальные пользователи ни за что не захотят терпеть постоянные ошибки, потери запросов, отказ в обслуживании и другие проблемы, связанные с нехваткой производительности сервиса. Они просто уйдут к конкурентам, даже если их предложение будет менее выгодным, но будет работать стабильно.

Самым очевидным решением проблемы может показаться увеличение мощности. Чем мощнее будет центральный сервер и чем больше у него будет ресурсов, тем меньше вероятность выхода из строя сервиса, или недоступность услуг. Да, это будет работать до определенного времени. До времени пока серверу не потребуется обслуживание.

Вторым решением будет сделать несколько серверов, которые в равной степени смогут поддерживать работоспособность сервиса предоставления услуг. Но при этом сразу же возникает несколько вопросов:

- Как клиентское приложение сможет определить сервер, который будет обслуживать клиента в данный момент времени?
- Как определить доступность сервера, и доступность сервиса в целом?
- Если один из серверов уже находится под нагрузкой из многочисленных запросов и не может позволить себе принять дополнительную нагрузку, то как определить, к какому доступному серверу необходимо обратиться, чтобы не создать второй загруженный до предела сервер?

Решением данной проблемы является балансировка нагрузки. Это совокупность методов распределения задач между устройствами в сети, с целью оптимизации используемых как аппаратных и вычислительных, так и сетевых ресурсов, сокращения времени обслуживания запросов и увеличения 'capacity'-максимального объема задач, которые может выполнить система [1].

Для решения данной задачи в компьютерной сети необходима разработка дополнительного программного обеспечения, которое бы взяло на себя функцию балансировщика. Балансировщик не обязательно может быть программным средством, но и отдельным сетевым устройством. На рис. 1 приведена принципиальная схема расположения балансировщика в клиент-серверной модели.

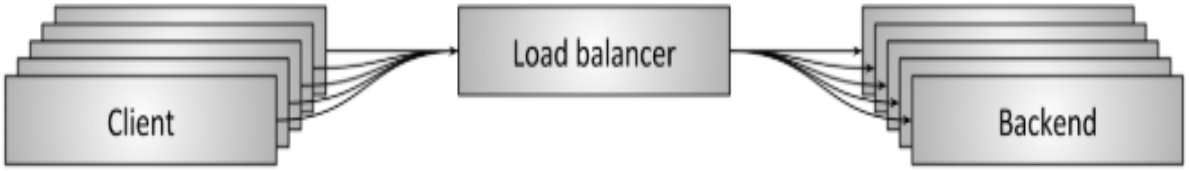

Рис. 1. Принципиальная схема расположения балансировщика
в клиент-серверной архитектуре

Введение третьего элемента в систему поможет избежать хранения на клиентской стороне о каждом сервере в системе, и сделать его более легким и производительным. Клиент может напрямую обращаться к балансировщику через заранее определенный механизм, а затем разрешить имя, согласно алгоритмам и политикам, заложенным в балансировщик.

Балансировщик сможет в фоновом режиме проводить мониторинг доступности серверов и маршрутов к ним, чтобы в момент запроса от клиента, маршрутизировать запросы вокруг перегруженного элемента сети, по самому короткому пути.

И самая важная функция балансировщика это устранение неоднородности в сети, потому что зачастую создать крупный сервис в одной зоне, или регионе недостаточно, и тогда на плечи балансировщика ложится преодоление подобных мелких системных нестыковок, которые для стандартной клиент-серверной модели существенно усложнили бы задачу [2].

Процедура балансировки осуществляется при помощи целого комплекса алгоритмов и методов, соответствующим следующим уровням модели OSI:

- сетевому;
- транспортному;

– прикладному.

Рассмотрим эти уровни более подробно.

Балансировка на сетевом уровне

Балансировка на сетевом уровне предполагает решение следующей задачи: нужно сделать так, чтобы за один конкретный IP-адрес сервера отвечали разные физические машины. Такая балансировка может осуществляться с помощью множества разнообразных способов.

- DNS-балансировка. На одно доменное имя выделяется несколько IP-адресов. Сервер, на который будет направлен клиентский запрос, обычно определяется с помощью алгоритма Round Robin.
- Построение NLB-кластера. При использовании этого способа серверы объединяются в кластер, состоящий из входных и вычислительных узлов.
- Распределение нагрузки осуществляется при помощи специального алгоритма. Используется в решениях от компании Microsoft.
- Балансировка по IP с использованием дополнительного маршрутизатора.
- Балансировка по территориальному признаку осуществляется путем размещения одинаковых сервисов с одинаковыми адресами в территориально различных регионах Интернета [2].

Балансировка на транспортном уровне

Этот вид балансировки является самым простым: клиент обращается к балансировщику, тот в свою очередь перенаправляет запрос одному из серверов, который и будет его обрабатывать. Выбор сервера, на котором будет обрабатываться запрос, может осуществляться в соответствии с самыми разными алгоритмами: путём простого кругового перебора, путем выбора наименее загруженного сервера из пула. Иногда балансировку на транспортном уровне сложно отличить от балансировки на сетевом уровне [3].

Различие между уровнями балансировки можно объяснить следующим образом. К сетевому уровню относятся решения, которые не терминируют на себе пользовательские сессии. Они просто перенаправляют трафик и не работают в проксирующем режиме.

На сетевом уровне балансировщик просто решает, на какой сервер передавать пакеты. Сессию с клиентом осуществляет сервер.

На транспортном уровне общение с клиентом замыкается на балансировщике, который работает как прокси-сервер. Он взаимодействует с серверами от своего имени, передавая информацию о клиенте в дополнительных данных и заголовках. Таким образом работает, например, популярный программный балансировщик HAProxy [4].

Балансировка на прикладном уровне

При балансировке на прикладном уровне балансировщик работает в режиме «умного прокси». Он анализирует клиентские запросы и перенаправляет их на разные серверы в зависимости от характера запрашиваемого контента.

Так работает, например, веб-сервер Nginx, распределяя запросы между фронтендом и бэкендом. За балансировку в Nginx отвечает модуль Upstream[1].

В качестве еще одного примера инструмента балансировки на прикладном уровне можно привести pgpool — промежуточный слой между клиентом и сервером СУБД PostgreSQL. С его помощью можно распределять запросы по серверам баз данных в зависимости от их содержания. Например, запросы на чтение будут передаваться на один сервер, а запросы на запись — на другой [5, 6].

На одном из проектов компании НетКрекер нам необходимо было решить задачу масштабирования сервиса и сокращения времени обработки запросов клиентов, находящихся в относительном удалении от серверов заказчика.

На тот момент на продакшн сервере использовалась архитектура, изображенная на рис. 2.

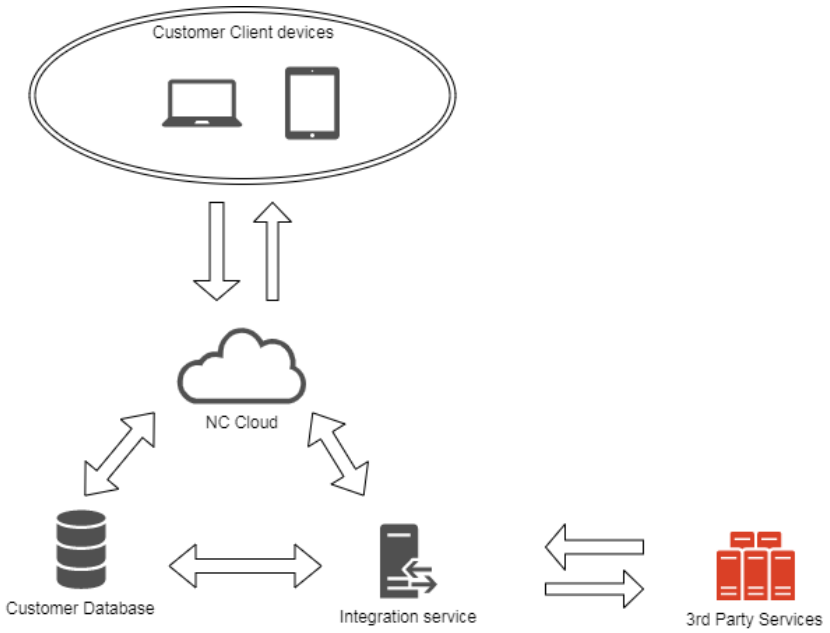

Рис. 2. Принципиальная архитектура имевшегося решения

И были проведены контрольные изменения скорости выполнения операций и результаты приведены в таблице 1.

Таблица 1.

Результаты контрольного измерения времени обработки операции

Операция	Заполнение текстового поля	Создание нового объекта	Асинхронная операция с участием сторонней системы	Обновление информации об объекте
Среднее время обработки (до внедрения балансировщика), секунд	2,1	4,6	3,8	1,5

[1] Recommendation ITU-T Y.1541(12/2011) Internet protocol aspects – Quality of service and network performance Network performance objectives for IP-based services (2011), p. 66.

Главной задачей являлось увеличение количества запросов, которые обрабатываются системой, поскольку заказчик планировал использовать данный сервис для расширения бизнеса в крупном городе на удалении более 1500 км.

Но для решения данной задачи имелось 2 существенных ограничения:

- База данных клиента находится в одной локации, и не может быть перемещена, скопирована или вынесена в облако.
- Сторонние сервисы имеют 1 сетевой адрес, и так же находятся в статичном положении.

Таким образом, мы получили довольно маленькую степень свободы, и нами были рассмотрены следующие варианты решения данной задачи.

1 Добавление нескольких Облачных элементов в архитектуру

На рис. 3 изображена принципиальная архитектура первой идеи, протестированной на тестовом окружении.

Основная ее идея состоит в организации облачного кластера при поддержке интеграционного сервиса в каждом новом регионе

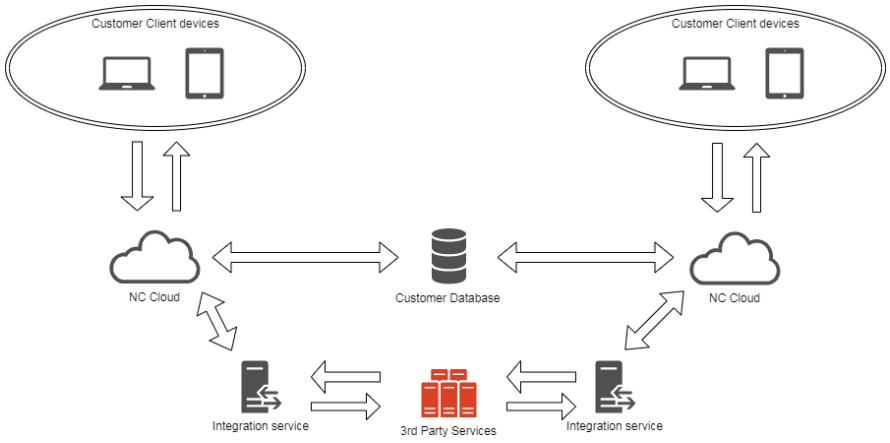

Рис. 3. Принципиальная архитектура первого из предложенных вариантов

По результатам контрольных измерений было установлено, что мы смогли ускорить выполнение Операций «Заполнение текстового поля» и «Обновление информации об объекте», так как эти операции не требовали прямых запросов в базу данных, и эта операция могла быть совершена внутри облака с локальной копией данных, и в последствии в фоновом режиме загружена в базу данных. А оставшиеся две исследуемые операции показали снижение скорости обработки, так как на используемые элементы архитектуры удвоилась нагрузка.

Но также обнаружились существенные недостатки:

- Возросло количество подключений к базе данных, а это повышает риск компрометирования данных, хранящихся в ней и как следствие, разглашение коммерческой тайны.
- Увеличение количество интеграционных сервисов замедлило работу сторонних систем, т. к. с их стороны не была проведена подготовка к подобному увеличению числа запросов.
- Облачные сервера оказались требовательны к ресурсу аппаратной части, и поэтому поддерживать большое количество облачных серверов не уложится в бюджет проекта.

2 Организация VPN тоннелей и балансировка нагрузки на компоненты

Основной идеей второго было введение в архитектуру балансировщика нагрузки и внедрение возможности динамически увеличивать производительность облачного кластера за счет внедрения резервирования дополнительных аппаратных ресурсов. Нагрузка между кластерами облака должна распределяться балансировщиком при помощи заложенных в него алгоритмов, а нагрузка на интеграционный сервис и базу данных регулируется собственными, более простыми балансировщиками, основной задачей которых является выравнивание потока поступающих на сервис заявок за счет буферизации

Для минимизации задержки сетевого соединения были организованы VPN тоннели между региональными агрегационными прокси-серверами и балансировщиком нагрузки.

На рис. 4 изображена принципиальная архитектура с использованием балансировщиков нагрузки.

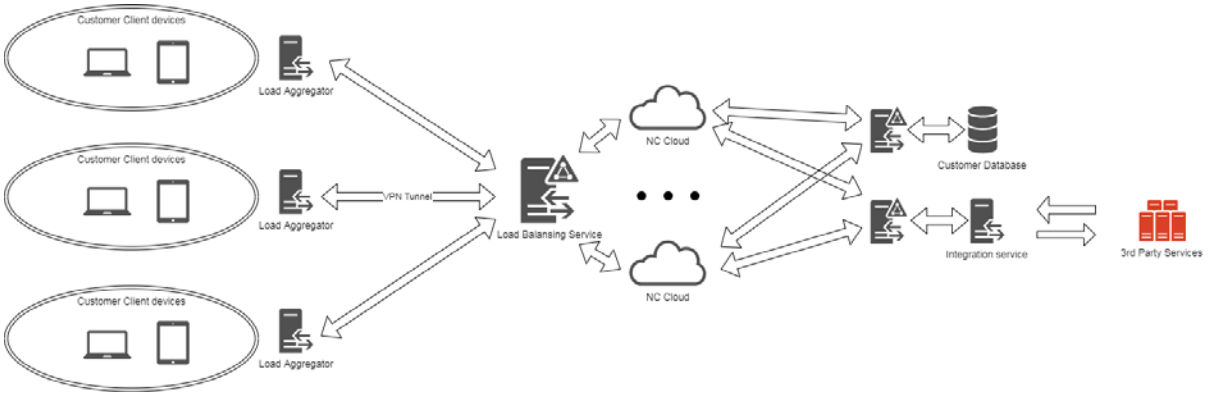

Рис. 4. Принципиальная архитектура с использованием балансировка нагрузки

По результатам контрольных измерений, было установлено, что все 4 исследуемые операции показали увеличение времени обработки только в моменты резкого увеличения потока заявок, поскольку балансировщик был настроен на мониторинг нагрузки на определенных временных отрезках. Результаты измерений приведены в таблице 2.

Таблица 2.

Результаты контрольного измерения времени обработки операции

Операция	Заполнение текстового поля	Создание нового объекта	Асинхронная операция с участием сторонней системы	Обновление информации об объекте
Среднее время обработки (после проведения работ по развертыванию новой архитектуры решения), секунд	2,2	5,0	3,8	1,7

Общее доля увеличения времени обработки операции составила примерно 10 %, что является приемлемым результатом для заказчика. Сводный график результатов из таблиц 1 и 2 приведен на рис. 5.

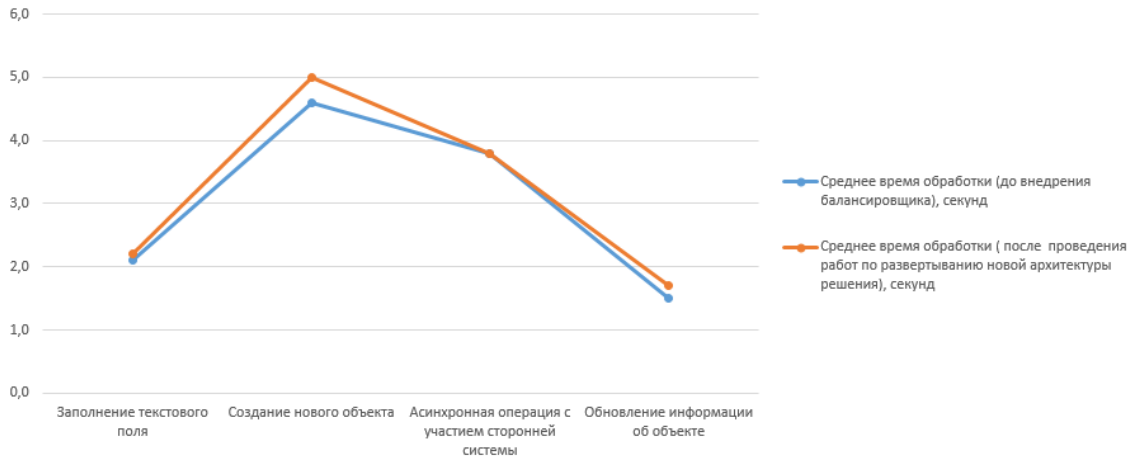

Рис. 5. Графики Контрольных измерений времени выполнения операций

Таким образом была решена задача увеличения объема обрабатываемых запросов, не используя пропорциональное увеличение аппартаных ресурсов, что позволило уложиться в заданый бюджет проекта. Данное решение поставленной задачи, так же оказалось простым в реализации, поскольку требует по большей части переработку программной части.

Заключение

Решение подобной задачи было бы невозможно без реализации балансировщика нагрузки в условиях имевшейся архитектуры решения. Добавление в архитектуру балансировщика нагрузки доказало все тезисы о роли балансировщика нагрузки, обозначенные в начале статьи, и позволило сделать SaaS-решение заказчика более гибким и устойчивым к нагрузкам.

Подобный подход был внедрен одним из первых, среди проектов компании, и получил положительный отзыв от клиентов заказчика. Это позволило нам рекомендовать внедрение балансировки нагрузки для микросервисных арихитектур коллегам на других проектах и сформировать best practice. Это набор советов, которые помогут коллегам обойти те ошибки, с которыми встретились мы, при внедрении балансирощика на проект.

Подводя итог, можно сказать, что, для высоконагруженных и динамически масштабируемых микросервисных архитектур, балансировщик является необходимым звеном в потоках передаваемых данных, от которого будет зависеть скорость обработки запроса, время отклика и количество принятых запросов. Эти три фактора напрямую влияют на оценку системы со стороны пользователя и таким образом помогут привлекать новых и удерживать существующих пользователей.

Литература

1. Елагин В. С., Онуфриенко А. В. Как оператору заработать на OTT-сервисах и при чем тут SDN? // T-COMM – Телекоммуникации и транспорт. 2017. Т. 11. № 1. С. 17–21.

2. Kotenko, A. Kuleshov and I. Ushakov, Aggregation of elastic stack instruments for collecting, storing and processing of security information and events // 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), San Francisco, CA, 2017, pp. 1–8.

3. Sharikov P. I., Krasov A. V., Ivanov A. V. Research of the possibilities of the technique of hidden embedding of a digital watermark in class-files on virtualized platforms with differing architecture // VestnikSPb of the State University of Emergencies Ministry of Russia. 2018. No. 2. pp. 79–88.

4. Гольдштейн Б. С., Соколов Н. А., Яновский Г. Г. Сети связи: учебник для вузов. СПб.: БХВ–Санкт-Петербург, 2010. 400 с.

5. Elagin V. S., Goldshtein A. B., Onufrienko A. V., Zarubin A. A., Belozertsev I. A. Synchronization of delay for OTT services in LTE // 2018 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO), Minsk, 2018, pp. 1–4.

6. Elagin V. S., Goldshtein B. S., Onufrienko A. V., Zarubin A. A., Savelieva A. A. The efficiency of the DPI system for identifying traffic and providing the quality of OTT services // 2018 Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow, Russia, 2018, pp. 1–5.

References

1. Elagin V. S., Onufrienko A. V. (2017). How to make money on the operator OTT-services, and what have the SDN? // T-Comm, vol. 11, no.1, pp. 17-20. (in Russian).

2. Kotenko, A. Kuleshov and I. Ushakov, Aggregation of elastic stack instruments for collecting, storing and processing of security information and events // 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), San Francisco, CA, 2017, pp. 1–8.

3. Sharikov P. I., Krasov A. V., Ivanov A. V. Research of the possibilities of the technique of hidden embedding of a digital watermark in class-files on virtualized platforms with differing architecture // VestnikSPb of the State University of Emergencies Ministry of Russia. 2018. No. 2. pp. 79–88.

4. Goldshteyn B. S., Sokolov N. A., Yanovskiy G. G. Seti svyazi: uchebnik dlya vuzov. SPb.: BKhV–Sankt-Peterburg. 2010. 400 s.

5. Elagin V. S., Goldshtein A. B., Onufrienko A. V., Zarubin A. A., Belozertsev I. A. Synchronization of delay for OTT services in LTE // 2018 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO), Minsk, 2018, pp. 1–4.

6. Elagin V. S., Goldshtein B. S., Onufrienko A. V., Zarubin A. A., Savelieva A. A. The efficiency of the DPI system for identifying traffic and providing the quality of OTT services // 2018 Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow, Russia, 2018, pp. 1–5.

Елагин Василий Сергеевич – кандидат технических наук, доцент Санкт-Петербургского государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича, ze_vs@outlook.com

Elagin Vasily – Candidate of Engineering Sciences, assistant professor, The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, ze_vs@outlook.com

Николаев Валентин Евгеньевич – студент Санкт-Петербургского государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича, eone_zed@inbox.ru

Nikolaev Valentin – Student, The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, eone_zed@inbox.ru