МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

(код и наименование направления подготовки)

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

(направленность (профиль))

# БАКАЛАВРСКАЯ РАБОТА

на тему «**УПРАВЛЕНИЕ ПОСЛЕДОВАТЕЛЬНОСТЬЮ ВИРТУАЛЬНЫХ ФУНКЦИЙ ПРОГРАММНО-ОПРЕДЕЛЯЕМОЙ СЕТИ SDN»**

| | | |
|---|---|---|
| Студент | Е.Ю. Лесных | |
| Руководитель | А.И. Туищев | |
| Консультант по аннотации | Н.В. Ященко | |

**Допустить к защите**

Заведующий кафедрой, к.тех.н., доцент, А. В. Очеповский _____

«_____» _____ 20___ г.

Тольятти 2017

# АННОТАЦИЯ

**Тема** выпускной квалификационной работы: «Управление последовательностью виртуальных функций программно-определяемой сети SDN».

Работа выполнена студенткой Тольяттинского Государственного Университета, института математики, физики и информационных технологий, группы ПМИб-1301, Лесных Екатериной Юрьевной. Выпускная квалификационная работа посвящена разработке программного кода для управления последовательностью виртуальных функций программно-определяемой сети (Software Defined Network, далее SDN).

**Объект** исследования - виртуальные функции программно-определяемой сети SDN.

**Предмет** исследования – определение последовательности виртуальных функций для оптимального расположения узлов в сетевом графе.

**Цель** ВКР – разработка модели оптимального построения сетевого графа последовательности виртуальных функций для стабильной работы программно-определяемой сети SDN.

Для достижения цели были поставлены следующие **задачи:**

- проанализировать учебную и научно-методическую литературу по предметной области, выявить особенности, достоинства и недостатки программно-определяемых сетей;
- исследовать существующую реализацию построения сетевого графа телекоммуникационной сети;
- представить математическую модель сетевого графа телекоммуникационной сети;
- разработать алгоритм для построения сетевого графа телекоммуникационной сети;
- минимизировать недостатки существующей реализации построения сетевого графа телекоммуникационной сети.

Выпускная квалификационная работа состоит из введения, двух глав, заключения, списка используемых источников. Ее можно разделить на несколько связанных частей. Большое внимание уделяется правильному построению сетевого графа и удобству использования программы. Первая глава описывает существующий алгоритм работы SDN, его минусы и возможные решения для их минимизации. Вторая глава – общее описание проблем сетевого графика, правила его построения, разработка программного кода. Эта часть описывает алгоритмы построения графов и выбора оптимального пути, тестирование разработанного кода и подведение итогов. В заключении производится общая оценка, анализ трудностей, возникших при разработке и проектировании, а также перспективы развития виртуальных функций SDN.

В результате, был разработан, описан и протестирован программный код для управления последовательностью виртуальных функций программно-определяемых сетей, а также был разработан алгоритм выбора оптимального графа.

Работа находится на стадии рассмотрения на внедрение.

Выпускная квалификационная работа содержит пояснительную записку объемом 51 страница, включает 20 иллюстрации, 3 таблицы, список литературы из 24 наименований, приложение.

# ANNOTATION

**The title** of the graduation work is “Managing the Sequence of Virtual Function of the Software Defined Network”. The work deals with the development of the program for lifecycle management of virtual functions of a Software Defined Network (SDN).

**The aim** of the work is to build a network graph for the stable operation of the SDN.

**The object** of the graduation work is Software Defined Network.

**The subject** of the graduation work is network graph used to build a network.

The graduation work may be divided into several connected parts. In the work much attention is given to correct construction of the network graph and the convenience of using the program. The first part describes the existing algorithm of SDN operation, its disadvantages and possible solutions for their minimization. The second part is a general part about the issues of network graph, building rules, development of program code. This part considers algorithms for constructing graphs and choosing the optimal path, test of developed code and summarizing.

As a result, the program code for managing the lifecycle of virtual function of the SDN was developed, described and tested. Algorithm for selecting the optimal graph was developed.

The graduation work consists of an explanatory note on 51 pages, including 20 figures, 3 tables, the list of 24 references, attachment.

**ОГЛАВЛЕНИЕ**

ВВЕДЕНИЕ ..... 3
Глава 1 ОПИСАНИЕ РЕШЕНИЯ ВИРТУАЛИЗАЦИИ СЕТЕВЫХ ФУНКЦИЙ ПРИ ПОМОЩИ SDN ..... 6
1.1 Общая характеристика деятельности организации ООО «Неткрэкер» 6
1.2 Существующая реализация построения графов сети ..... 8
1.3 Постановка задачи построения оптимального сетевого графа ..... 14
1.4 Реализация виртуализации сетевых функций различными компаниями ..... 25
Глава 2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПОСТРОЕНИЯ СЕТЕВЫХ ГРАФОВ ПОСЛЕДОВАТЕЛЬНОСТИ ВИРТУАЛЬНЫХ ФУНКЦИЙ ..... 29
2.1 Проектирование разрабатываемого программного модуля для построения сетевых графов последовательности виртуальных функций ..... 29
2.1.1 Обоснование архитектуры разрабатываемого программного модуля построения сетевых графов ..... 29
2.1.2 Обоснование выбора средств реализации разрабатываемого программного модуля для построения сетевых графов ..... 32
2.2 Реализация программного модуля для построения оптимального сетевого графа последовательности виртуальных функций ..... 36
2.3 Описание примера работы программного модуля построения оптимального сетевого графа ..... 38
2.4 Тестирование разработанного программного модуля для построения оптимального сетевого графа ..... 41
ЗАКЛЮЧЕНИЕ ..... 45
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ ..... 47
ПРИЛОЖЕНИЕ А Листинг кода для определения кратчайшего пути с учетом длины ребра ..... 50

# ВВЕДЕНИЕ

Жизнь современного человека не может не включать в себя использование интернета. С его помощью можно записаться на прием к врачу, оплатить штраф или коммунальные услуги, найти нужную информацию, проложить путь в незнакомой местности и так далее.

В настоящее время телекоммуникационные компании по предоставлению услуг связи – это неотъемлемая часть жизни любого человека, фирмы и даже страны.

Пользователей интернета с каждым годом становится все больше [13] (рисунок 1), поэтому ресурсы телекоммуникационных компаний должны увеличиваться непрерывно, чтобы обеспечить связь по всему миру.

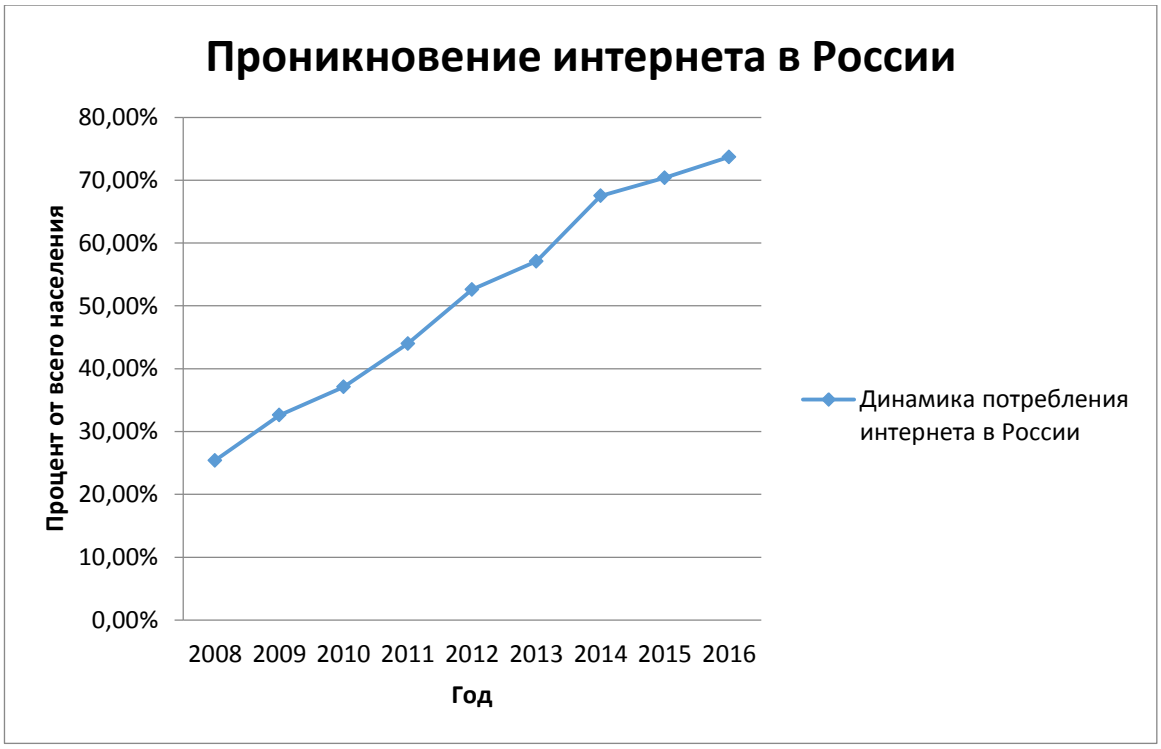


Рисунок 1 - Проникновение Интернета в России

Каждый оператор сети имеет множество разнообразного оборудования, количество которого увеличивается с каждым годом. Для расширения сети ему необходимо запустить новое оборудование в специализированном помещении с достаточным энергопитанием. Все это стоит больших денег: возрастает энергопотребление, появляется необходимость в найме квалифицированного

персонала, и в капитальных, операционных затратах. Также аппаратные сетевые устройства имеют свойства устаревать, их функционал становится недостаточным, что приводит к повторению циклов «закупка – проектирование – интеграция – развертывание». Это не увеличивает доходов оператора сети, а скорее приводит к тому, что расходы на построение сети постепенно начинают опережать доходы, так как срок службы оборудования с каждым годом становится меньше [16].

Такой путь развития операторских сетей устарел, нужны свежие взгляды на бизнес сервис-провайдеров и операторов сети. Одно из решений данной проблемы – это виртуализация сетевых функций - Network Functions Virtualization (NFV), связанная с концепцией программно-конфигурируемых сетей - Software Defined Network (SDN) [13].

Необходимость экономить средства на покупке нового оборудования определяет актуальность работы.

**Объект** исследования - виртуальные функции программно-определяемой сети SDN.

**Предмет** исследования – определение последовательности виртуальных функций для оптимального расположения узлов в сетевом графе.

**Цель** ВКР – разработка модели оптимального построения сетевого графа последовательности виртуальных функций для стабильной работы программно-определяемой сети SDN.

**Задачами** работы являются:

- проанализировать учебную и научно-методическую литературу по предметной области, выявить особенности, достоинства и недостатки программно-определяемых сетей;
- исследовать существующую реализацию построения сетевого графа телекоммуникационной сети;
- представить математическую модель сетевого графа телекоммуникационной сети;

- разработать алгоритм для построения сетевого графа телекоммуникационной сети;
- минимизировать недостатки существующей реализации построения сетевого графа телекоммуникационной сети.

Бакалаврскую работу можно разделить на несколько логически связанных частей. Большое внимание уделяется правильному построению сетевого графа и удобству использования программы.

В первой главе дается описание предметной области, описывается существующий алгоритм работы SDN, его достоинства, недостатки и возможные решения для их минимизации. Определяются требования для новой технологии, выбираются оптимальные алгоритмы построения сетевых графов.

Во второй главе представлено обоснование выбора технологий для разработки программного модуля, описывается кодирование, приводятся блок-схемы реализованного программного модуля. Так же в главе описано тестирование программного кода, подведены итоги.

В результате выполнения ВКР, был разработан, описан и протестирован программный код для управления последовательностью виртуальных функций программно-определяемых сетей.

# Глава 1 ОПИСАНИЕ РЕШЕНИЯ ВИРТУАЛИЗАЦИИ СЕТЕВЫХ ФУНКЦИЙ ПРИ ПОМОЩИ SDN

## 1.1 Общая характеристика деятельности организации ООО «Неткрэкер»

Заказчиком разработки программного модуля для управления последовательностью виртуальных функций программно-определяемой сети является ООО «Неткрэкер». Одно из направлений, которым занимается компания – это создание, внедрение и сопровождение SDN и NFV решений для операторов связи.

Неткрэкер является дочерней компанией корпорации NEC. Центры разработки, технической поддержки и отделы продаж расположены по всему миру: Россия, Украина, Белоруссия, США, Индия. Также компания имеет свои офисы в Северной Америке, Европе, Южной Африке, Латинской Америке, на Ближнем Востоке и в Азиатско-Тихоокеанском регионе [9].

24 февраля 2015 года корпорация NEC и Неткрэкер объявили о совместной инициативе по реализации возможностей виртуализации сетевых решений (SDN) и сетевых функций (NFV) для различных сфер бизнеса. Виртуализация сети и инфраструктуры для операторов связи предлагает беспрецедентную возможность кардинально улучшить качество работы и предлагать новые услуги [11].

В результате данной инициативы был создан новый глобальный бренд и структура бизнеса, объединяющая в себе лучшие из сетевых инноваций NEC и многолетний опыт IT-лидера Неткрэкер в области телекоммуникаций для предоставления комплексных решений SDN / NFV в многодоменной среде. Новый бизнес-бренд получил название «NEC / Netcracker SDN / NFV Solutions» [12].

NEC работает над созданием и предоставлением простой и гибкой сетевой инфраструктуры нового поколения, которая будет обеспечивать внедрение и применение виртуализации для операторских сетей. Эта инфраструктура будет использовать SDN / NFV, как стратегически приоритетную область. В 2011 году NEC выпустил первый в мире контроллер /

коммутатор OpenFlow, который предлагает решения SDN / NFV, в том числе виртуализованное расширенное пакетное ядро (vEPC), виртуализованное оборудование для помещений клиентов (vCPE) и виртуальные мобильные операторы виртуальной сети (vMVNO) с рядом коммерческих функций в тестовом режиме [10].

Неткрэкер признан лидером рынка в отрасли BSS / OSS, предоставляя передовые телекоммуникационные системы и системы управления для операторов CSP и кабельных операторов всех масштабов по всему миру. Неткрэкер также является лидером в разработке решений для управления и оркестровки (MANO) для виртуализации [13].

Совместная инициатива «NEC / Netcracker SDN / NFV Solutions» представляет собой уникальный подход, который использует глобальный масштаб двух компаний, известных своими инновациями в сетях и лидирующими позициями на рынке IT-решений. Их разнообразные знания в области сетевого оборудования, CPE, сетевого управления, решений OSS и оркестровки позволяют NEC и Неткрэкер объединить лучшие сетевые и IT - экспертные знания и опыт в виртуальных и традиционных сетях [9].

Новый глобальный бренд и структура бизнеса «NEC / Netcracker SDN / NFV Solutions» имеет специализированные бизнес-структуры по всем ключевым функциям, включая исследования и разработку, продажи и маркетинг [11].

Совместная разработка решений NEC и Неткрэкер будет укрепляться и углублять существующую координацию, согласовывать свои усилия в области развития с целью вывода на рынок комплексных решений SDN / NFV [11].

«SDN / NFV обладает потенциалом для радикального изменения операций и управления сетью операторов. Решения NEC и Неткрэкер в этой области получают высокие оценки от клиентов» [13], - сказал Шуничиро Теджима, исполнительный вице-президент корпорации NEC. «Новая инициатива была создана, чтобы интегрировать сильные стороны обеих компаний и предложить большую клиентскую ценность. В рамках нового

бренда мы будем управлять бизнесом SDN / NFV и вносить вклад в виртуализацию инфраструктуры и сетей» [12].

SDN / NFV направления в компании Неткрэкер активно развиваются, непрерывно идет работа над устранением ошибок и над разработкой нового функционала [9].

Резюмируя выше сказанное, можно сделать вывод, что SDN / NFV направление является перспективным, компания заинтересована в его развитии и всячески способствует улучшению существующей реализации. Для разработки корректных алгоритмов построения графов сети, преступим к анализу существующей реализации для выявления достоинств и недостатков.

## 1.2 Существующая реализация построения графов сети

Разработкой архитектуры NFV MANO (Network Function Virtualization, далее NFV, Management and Orchestration, далее MANO) занимается специальная группа Европейского Института Телекоммуникационных Стандартов – ETSI NFV Industry Specification Group (ISG). Она устанавливает стандарты управления и оркестрации всех ресурсов NFV в рамках облачных центров обработки данных. Эти ресурсы включают в себя вычислительные, сетевые ресурсы, системы хранения данных, виртуальные машины и др. Данный стандарт используется для реализации MANO в Неткрэкере [9].

Цель MANO – создание инфраструктуры для гибкого развертывания и управления сетевыми функциями, а также упорядочение хаоса, который может возникнуть в связи с быстрым темпом появления на рынке новых виртуальных сетевых функций [13].

Виртуализация сетевых функций – это технология виртуализации физических сетевых элементов телекоммуникационной сети, когда сетевые функции исполняются программными модулями, работающие на стандартных серверах (чаще всего x86) и виртуальных машинах (Virtual Machine, далее VM) в них. Эти программные модули могут взаимодействовать между собой для

предоставления услуг связи, что раньше осуществлялось аппаратными платформами [13].

NFV использует стандартную технологию виртуализации, чтобы обеспечить быстрое внедрение услуг для сетевых операторов. Большинство современных сетей состоят из разнообразных сетевых устройств, которые подключены или связаны друг с другом определенным образом для достижения желаемой функциональности сетевого сервиса. NFV ставит своей целью заменить эти сетевые устройства виртуализированными сетевыми функциями, которые могут быть объединены с промышленными серверами, коммутаторами и хранилищами большого объема. Они могут быть расположены в центрах обработки данных, сетевых узлах или в помещениях конечного пользователя. Эти виртуальные сетевые функции могут затем комбинироваться с использованием динамических методов для создания и управления сетевыми службами в режиме реального времени [10].

В NFV свойствах, отношения и другие метаданные подключений указаны в абстракциях виртуальных ссылок. Для моделирования того, как виртуальные каналы подключаются к виртуальным сетевым функциям, NFV использует точки подключения (Connection Point, далее CP), которые представляют виртуальные и / или физические интерфейсы виртуальных сетевых функций (Virtual Network Function, далее VNF) и связанные с ними свойства, а также другие метаданные. Элементы сети расположены в сетевой службе (Network Service, далее NS). NS - это набор сетевых функций, который определяет функции и поведенческую спецификацию. Следовательно, NS (рисунок 2) можно рассматривать архитектурно, как график пересылки сетевых функций (Network Functions, далее NF), связанных между собой посредством поддержки сетевой инфраструктуры [12].

Топология сетевых подключений касается только того, как подключаются разные VNF и как данные проходят через эти соединения, независимо от местоположения и размещения базовых физических сетевых элементов. График

пересылки сети определяет последовательность VNF, которые должны быть пересечены набором пакетов, соответствующих определенным критериям.

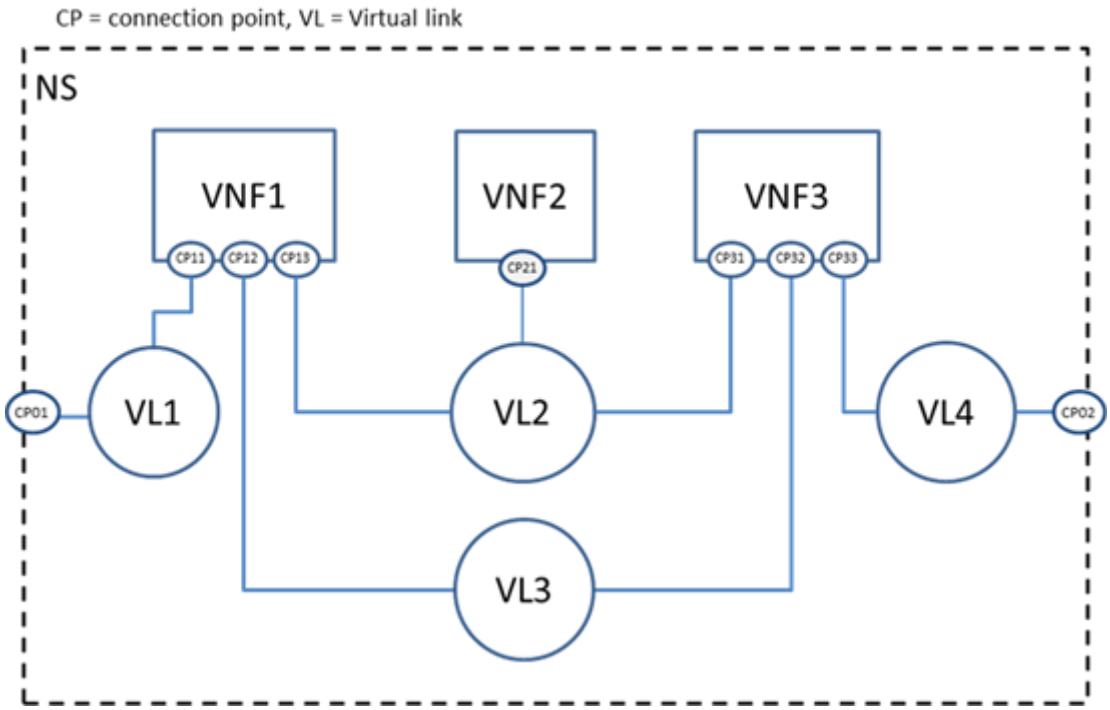


Рисунок 2 - Структура NS

Элементы схемы:

- NS (Network Service) - это набор сетевых функций;
- VNF (Virtual Network Function) – виртуальная сетевая функция, например, коммутатор, маршрутизатор, Wi-Fi роутер;
- CP (Connection Point) - виртуальные и / или физические интерфейсы;
- VL (Virtual Link) - логическая (виртуальная) локальная компьютерная сеть.

Граф пересылки сети, который строится NS-оператором, включает в себя критерии, определяющие, какие пакеты следует маршрутизировать через график. Простым примером этого может быть маршрутизация на основе адресов источника или ряда других различных приложений. Разные графики пересылки могут быть построены в одной сетевой топологии на основе различных критериев.

На рисунке 3 показан пример трех графиков пересылки VNF, установленных поверх топологии сетевых подключений, описанной ранее [10].

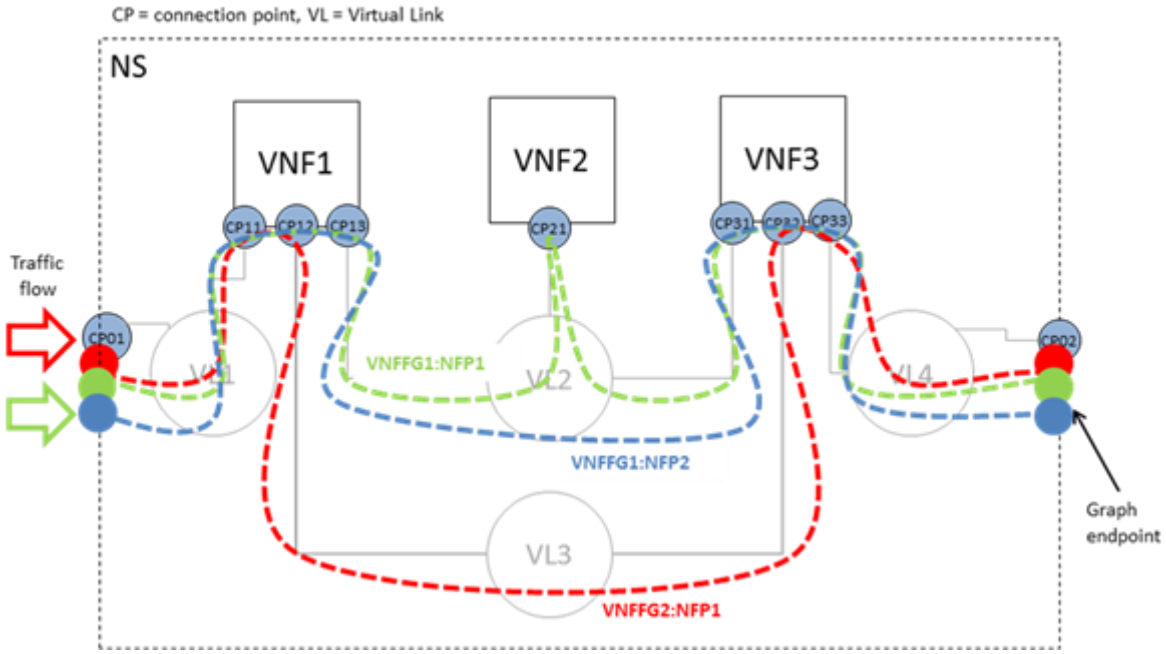


Рисунок 3 - Структура NS с возможными графами

Под сетевым графиком (сетевым графом) понимается абстрактный математический объект, представляющий из себя множество вершин графа (VNFs) и набор рёбер, то есть соединений между парами вершин.

Вершины сетевого графа выбираются в зависимости от того, какие сетевые функции выбрал пользователь. Например, на рисунке 3 представлены три варианта сетевых графиков:

- граф, проходящий через VNF#1, VNF#2 без возможности их администрирования (без включения точки подключения, отвечающей за настройку VNF);
- граф, проходящий через VNF#1, VNF#2 с возможностью их администрирования (с включением точки подключения, отвечающей за настройку VNF);
- граф, проходящий через VNF#1, VNF#2, VNF#3.

Рассмотрим архитектуру SDN (рисунок 4), так как разрабатываемый программный модуль будет встроен в нее.

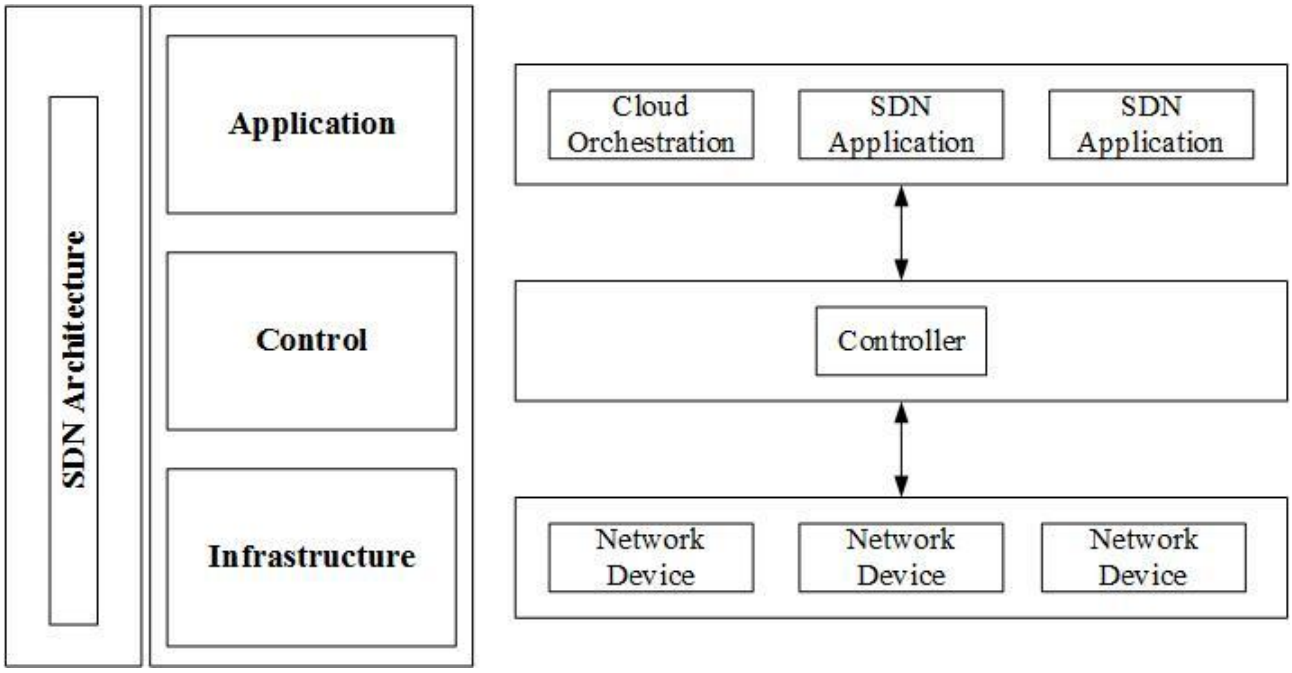


Рисунок 4 - Архитектура SDN

На рисунке 4 представлена архитектура SDN, на которой выделено три уровня:

- Application – представляет собой открытые программируемые интерфейсы для организации автоматизированной сетевой службы;
- Control – это отдельный блок управления и данных; абстрактная плоскость управления устройств;
- Infrastructure – стандартный доступ к сетевым устройствам.

Кроме классического управления сетью прямыми командами системного администратора к контроллеру, SDN контроллер поддерживает запуск на себе приложений управления сетью.

Каждое SDN приложение являет собой интерфейс оптимизации сети под конкретное бизнес приложение и его основная роль — изменение сети в реальном времени под текущие нужды обслуживаемой программы. На данный момент не реализовано SDN приложение, способное строить оптимальный сетевой граф, поэтому выполнение этих обязанностей лежит на NS-операторе. Рассмотрим существующий алгоритм создания сетевого графа.

Существующий алгоритм создания сетевого графа можно представить в виде диаграммы деятельности NS-оператора (рисунок 5). В настоящий момент вся настройка взаимодействия сетевых элементов лежит на NS-операторе. Он должен создать VNFs, произвести их настройку, создать необходимые CPs и виртуальные связи (Virtual link, далее VL), которые описывают базовую топологию подключений, а также другие необходимые параметры [11].


Рисунок 5 - Диаграмма деятельности NS-оператора

Недостатки данного алгоритма:

- построение сетевого графа происходит в ручном режиме;

- не строятся альтернативны графы;
- не выбирается оптимальный путь;
- при большом количестве виртуальных функций, редактирование сетевого графа занимает большое количество времени.

Исходя из найденных недостатков, видно, что процесс создания сетевых графов нуждается в автоматизации и может быть упрощен. В следующем пункте рассмотрим новое решение построения сетевого графика последовательности виртуальных функций, которое предполагает применение программного модуля, направленного на его автоматическое построение.

### 1.3 Постановка задачи построения оптимального сетевого графа

Целью работы является разработка модели оптимального построения сетевого графа последовательности виртуальных функций для стабильной работы программно-определяемой сети SDN. Для достижения цели, предлагается строить сетевые графы автоматически. Такое решение устранит недостатки существующего алгоритма построения сетевых графов и реализует следующие возможности:

- программное построение сетевого графа;
- выбор оптимального пути в графе;

Полностью снять задачу построения сетевых графов с NS-оператора не представляется возможным, так как часто возникают ситуации, в которых последовательность связи VNFs имеет значение.

Опишем новый алгоритм деятельности NS-оператора:

- создание виртуальных сетевых функции;
- настройка виртуальных сетевых функции – создание CPs;
- прокладка путей между VNFs, в тех случаях, когда это необходимо.

Если же порядок взаимодействия виртуальных сетевых функций не имеет значения, то граф полностью строится автоматически. Разработанный алгоритм автоматизации построения сетевого графа представлен на рисунке 6.


Рисунок 6 - Диаграмма деятельности NS-оператора

**Постановка задачи:** требуется решить задачу маршрутизации трафика, которая заключается в следующем: имеется конечное множество узлов (VNFs) сетевого графа, известно время отклика между узлами в текущий момент времени. Необходимо найти оптимальный путь пересылки трафика, с минимальными временными затратами.

При решении задачи маршрутизации необходимо учитывать следующие ограничения:

- трафик не должен проходить через одну вершину более одного раза;

- трафик должен пройти через все вершины.

Рассмотрим основные определения, которые будут использоваться в работе.

Нагруженным неориентированным графом $G = (V, E, U)$ называется тройка множеств:

- $V = (v_1, v_2, \dots, v_n)$ – множество вершин, числом вершин будем считать $V = n$;
- $E \subseteq V \times V$ – множество ребер графа $v_i, v_j = e\ i,\ j$ , будем считать, что $E = m$ – количество ребер графа.
- $U: E \to 0, +\infty$ - весовая функция на ребрах, весом ребра обозначим $u\ v_i, v_j$ между вершинами $v_i$ и $v_j$. Для вершин, не связанных ребром, положим $u\ v_i, v_j = \infty$.

Нагруженный граф будем задавать с помощью матриц длин ребер.

Если $v_i, v_j \in E$, то вершины $v_i, v_j$ называются смежными.

Ребро $e(i, j)$ инцидентно вершинам $v_i$ и $v_j$, если $v_i, v_j \in V$ и $e\ i, j \in E$.

Количество ребер графа, инцидентных $v_i$ называют степенью $d(v_i)$ вершины $v_i$.

Если справедливо $m \ll n^2$, то граф называется разреженным.

Если любая пара вершин соединена одним ребром, то такой граф называют полным.

Путь в графе – чередующаяся последовательность вершин и ребер $_{i_o, j_k} = (v_{i_0}, v_{j_k}) = v_{i_0}, e\ i_0, i_1\ , v_{i_1}, \dots, v_{i_{k-1}}, e\ i_{k-1}, i_k\ , v_{i_k}$ , каждое ребро инцидентно двум вершинам – непосредственно предшествующей ему и непосредственно следующей за ним.

Сумма весов входящих в путь ребер – это длина пути.

Кратчайшим путем $(v_i, v_j)$ между вершинами $v_i$ и $v_j$ называют путь, имеющий минимальную длину между вершинами $v_i$ и $v_j$.

Длину кратчайшего пути между $v_i$ и $v_j$ $(m\ v_i, v_j = 0, i = 1, \dots, n)$ называют расстоянием $m\ v_i, v_j$ между вершинами $v_i$ и $v_j$.

Если между любыми двумя вершинами графа существует путь, то есть $m\ v_i, v_j\ < \infty$ для всех $i, j$, то граф называют связанным.

Графы, не имеющие петель и кратных ребер, называются простыми.

Между начальной и конечной вершинами графа может существовать несколько кратчайших путей равной длины. Это обстоятельство в данной работе не является существенным, поэтому при всех упоминаниях кратчайшего пути будем подразумевать любой кратчайший путь.

Пусть $(v_i, v_j)$ – кратчайший путь в графе $G$. Тогда любой его подпуть тоже является кратчайшим.

Классическая задача о кратчайшем пути (shortest path problem, SP) [21] в наиболее общем виде формулируется как «найти пути между элементами двух множеств вершин $V_1$ и $V_2$ графа так, чтобы длины найденных путей являлись минимальными в данном графе между соответствующими вершинами».

Задача о кратчайшем пути является одной из важнейших в алгоритмической теории графов.

Различают два варианта задач MDSP (Multiple-destination shortest paths):

- нестационарная задача – вес ребра $e\ i, j$ равен времени перемещения из $v_i$ в $v_j$, которое задается предопределенной функцией времени, т.е. вес ребра $e\ i, j$ в некотором пути зависит от времени начала обхода пути и от времени, потраченного на обход ребер, предшествующих $e\ i, j$;
- вариант задачи, при котором граф меняется через определенные промежутки времени, являясь статическим между этими промежутками. Второй вариант задачи может иметь две формулировки: частично динамическая задача (можно либо только удалять ребра, либо только добавлять), полностью динамическая задача (разрешены и удаление, и вставка ребер). В обеих формулировках разрешена операция изменения веса ребра setWeight(i, j, w) .

Нестационарная задача DSP с применением техник для задачи SP разрешима за полиномиальное время на графах, обладающих свойством FIFO

[2], и NP-трудна на графах этим свойством не обладающих [2]. Задачи DSP с периодическим изменением графа могут быть решены сведением к соответствующим задачам и вычислением кратчайших путей графа «с нуля» при каждой необходимости или с использованием техники реоптимизации кратчайших путей [2].

Решение классической задачи SP на полученном для данного графа $G$ спаннере (spanner) H. Говорят, что граф $H = V, E^s : E^s \subseteq E\ (\alpha, \delta)$ - спаннер графа $G = V, E$ , если $\forall v_i, v_j \in V$ справедливо $m^h\ v_i, v_j \leq \alpha \cdot m^g\ v_i, v_j + \delta$, где $m^h\ v_i, v_j$ , $m^g\ v_i, v_j$ – кратчайшие расстояния между вершинами $v_i, v_j$в графах H и $G$ соответственно.

В простейшем случае, когда рассматривается задача SSSP, классическим алгоритмом поиска решения считается поиск в ширину (breadth-first search, BFS) [2]. BFS может решать задачу как на неориентированных, так и на ориентированных графах. Принцип работы алгоритма состоит в обходе вершин графа в порядке обнаружения и выполнение процесса релаксации для дуг между текущей вершиной и каждой новой обнаруженной. Процесс релаксации дуг – уточнение (уменьшение), если это возможно, оценки длины пути от исходной вершины до новой обнаруженной проведением пути через текущую вершину. BFS используется как основа алгоритмов для многих других задач на графах и имеет временную оценку сложности $O(m + n)$, являясь быстрейшим для SSSP на ненагруженных графах.

Существует также ряд важных алгоритмов для неклассических задач о кратчайшем пути. Алгоритм $A^*(A\ star)$ [6] – информированный поисковый алгоритм, используемый для поиска кратчайших путей из исходной вершины $s$, если для графа известна дополнительная информация. Обход вершин в алгоритме производится методом BFS, а порядок обхода вершин определяется эвристической функцией $f\ x = g\ x + h(x)$ , где $g\ x$ – известное расстояние от $s$ до $x$ , а $h(x)$ – эвристическая оценка расстояния от $s$ до $x$, которая должна быть допустимой эвристикой [6].

При решении задачи SPSP, могут применяться двунаправленные версии алгоритмов, используемых для поиска путей ($A^*$, BFS, Дейкстры). При двунаправленном поиске [7] выполняется одновременный поиск из исходной вершины $s$ и вершины назначения $t$. Асимптотическая сложность двунаправленного поиска определяется через коэффициент ветвления $b$ [7] как $O\ b^{\frac{d}{2}} + O\ b^{\frac{d}{2}}$ относительно сложности $O\ b^d$ однонаправленной версии поиска.

Если граф является ориентированным ациклическим (DAG), решение задачи SSSP на нем может быть получено за время $O(m+n)$ [2]. Сначала вершины графа сортируются в топологическом порядке, после производится ослабление ребер, выходящих из каждой вершины.

Такого рода алгоритмы основываются на понятие об иерархии вершин – кратчайшие пути между сильно удаленными друг от друга вершинами графа чаще проходят через «более важные вершины» (highway nodes), чем через остальные. Соответственно, такие алгоритмы сочетают в себя способ построения иерархии и алгоритм, используемый для поиска путей на получившейся иерархии.

Используя алгоритм поиска в ширину BFS, определим величину длины дуги:

$$m = \min s_i,\ i \in\ 1; N\ , \tag{1.1}$$

где $s_i$ – величина, вычисляемая по формуле:

$$s_i = \ \sum_{j=1}^{N-1} q_i \cdot t_i, \tag{1.2}$$

где $q_{ij} = \{0,1\}$ – булева функция, значение которой задается в зависимости от наличия отклика вершины $v_{j+1}$ на сигнал от $v_j$для -го графа,

$t_{ij}$ – время выполнения вершиной $v_j$ тестовой команды.

Определим алгоритм для нахождения оптимального сетевого графа последовательности виртуальных функций (рисунок 7):

1. Рекурсивно, начиная с вершины «Primary Inputs» (от входов схемы у выходам), для всех смежных вершин произведем естественное соединение соответствующих отношений и вычислим значение $s_i$ для конечной вершины.

2. Рекурсивно, начиная с вершины «Primary Outputs» (от выхода схемы к входам), для всех смежных вершин произведем естественное соединение соответствующих отношений, и спроецируем результат на начальную вершину.


Рисунок 7 - Алгоритм решения задачи

Описав алгоритм нахождения оптимального сетевого графа последовательности виртуальных функций, рассмотрим пример для четырех вершин. Пусть, VNF#1 – начальная вершина. (рисунок 8).

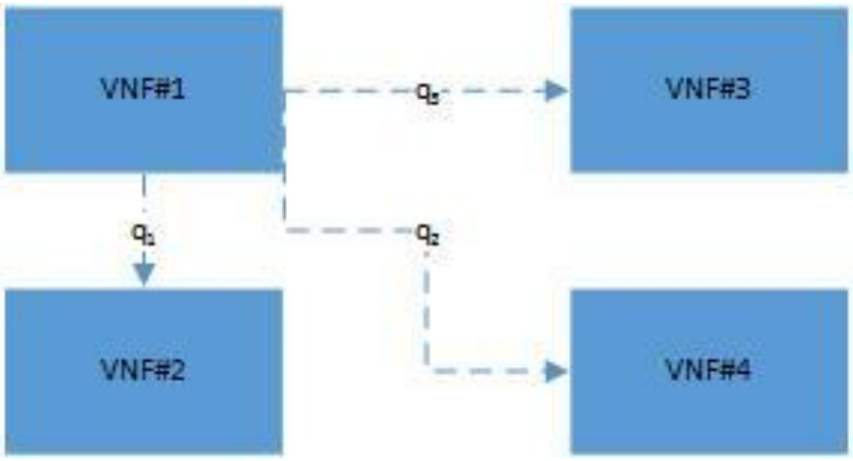


Рисунок 8 - Пример построения графа сети

Приняв VNF#1 за начальную вершину, посылаем от нее запросы на все созданные VNFs, вычисляя значение $s_i$ по формуле (1.2). Из найденных $s_i$ выбираем минимальное значение и принимаем $i-$ый путь за ребро графа.

Пусть

$$s_1 = \min s_i, i \in 1, N-1 \quad (1.3)$$

На следующем шаге также вычисляя значение $s_i$ для всех узлов, исключая те, которые уже соединены ребром, выбранным на предыдущем шаге (рисунок 9).

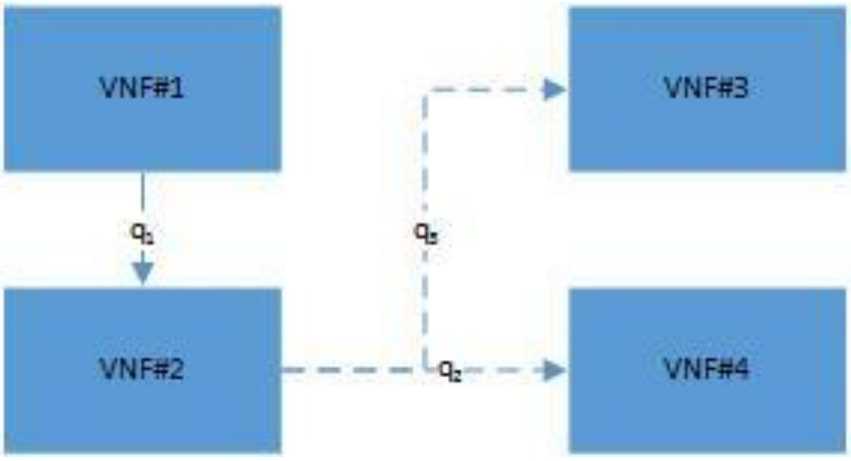


Рисунок 9 - Пример построения графа сети

Пусть

$$s_2 = \min s_i, i \in \ 2, N-1 \ , \qquad (1.4)$$

тогда за ребро графа принимаем путь от VNF#2 к VNF#3 и переходим к следующему шагу (рисунок 10).

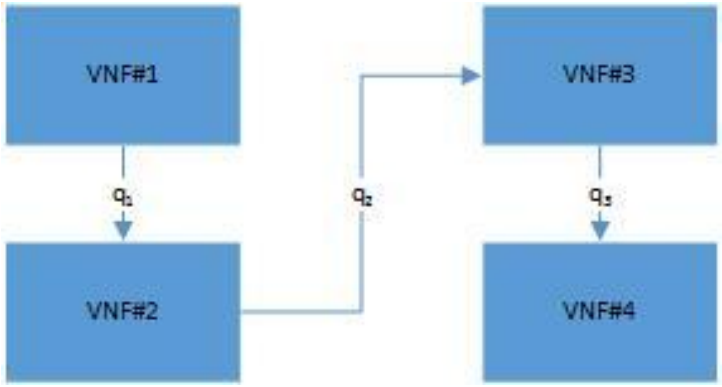


Рисунок 10 - Пример построения графа сети

Так как от VNF#3 до VNF#4 возможно проложить только один путь, принимаем его за ребро графа.

Полученный граф является графом с кратчайшим путем с начальной вершиной VNF#1.

Рекурсивно повторяя описанную последовательность действий, найдем 4 графа с кратчайшим путем (рисунок 11).

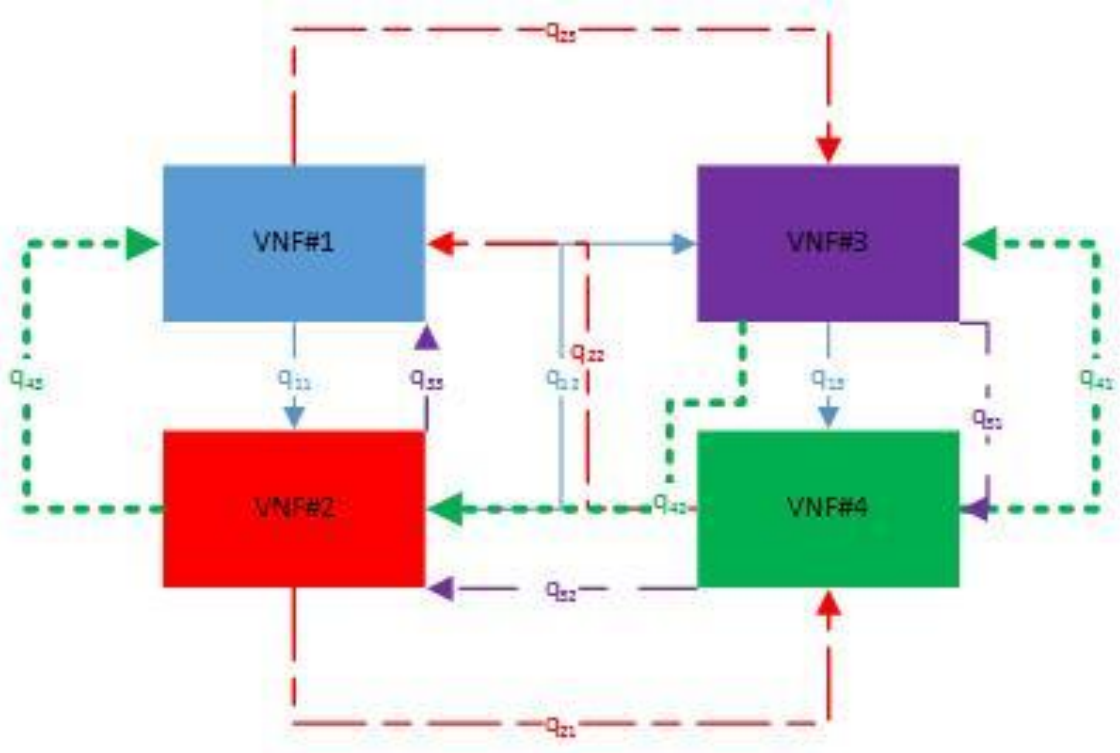


Рисунок 11 - Построенные графы

Сравнив полученные значения $s_i$, находим граф с кратчайшим путем.

$$S = \min s_i,\ i\epsilon\ 1; N\ . \quad (1.5)$$

Для реализации построения графов рассмотрим 2 способа их представления: с помощью матрицы расстояний и с помощью списка смежности.

Матрица расстояний (таблица 1) является удобным способом представления плотных графов, в которых количество ребер примерно равно количеству вершин в квадрате. В данном представлении заполняется матрица размером M на M, где M – количество вершин, помещая в ячейку A[i][j] значение расстояния между i-ой и j-ой вершинами:

Таблица 1 - Матрица расстояний

| | VNF#1 | VNF#2 | VNF#3 |
|---|---|---|---|
| VNF#1 | 0 | 20 | 37 |
| VNF#2 | 20 | 0 | 13 |
| VNF#3 | 37 | 13 | 0 |

Матрица расстояний - это способ представления графов, подходящий для ориентированных и неориентированных графов. Неориентированные графы представляются в виде симметричной матрицы A, так как, если существует ребро между i и j, то оно является и ребром из i в j, и ребром из j в i. Благодаря этому свойству можно сократить почти в два раза использование памяти, храня элементы только в верхней части матрицы, над главной диагональю [17].

С помощью данного способа представления, можно быстро проверить существует ли ребро между вершинами v и u, просто посмотрев в ячейку A[v][u] [17].

Другим способом представления графов являются списки смежности. Этот способ представления больше подходит для разреженных графов, то есть графов, у которых количество ребер гораздо меньше, чем количество вершин в квадрате. В данном представлении используется двумерный массив A содержащий M списков. В каждом списке A[i] содержатся все вершины u, так

что между v и u есть ребро. Память, требуемая для представления, равна M + N, где M – количество вершин, а N – количество ребер графа.

Сравнив способы представления графов, делаем вывод, что для описанной предметной области больше подходят списки смежности, так как в графах, которые предстоит строить, количество ребер гораздо меньше, чем количество вершин в квадрате, следовательно, нет необходимости использовать таблицы смежности, которые занимают значительно больше памяти.

Результатом решения задачи, поставленной выше, на графе $G$ является матрица расстояний $M_r$. Элементы матрицы расстояний $M$ для вершин $v_i, v_j$ из графа $G$ изменяются по формуле $m\ v_i, v_j = m^r(v_i, v_j)$ и производится изменение матрицы последовательностей $P$ по формулам:

$$p\ v_i, v_j = \begin{matrix} p^r\ v_i, v_j \text{ , если } m\ v_i, v_j > m^r\ v_i, v_j \text{ и } p\ v_i, p^r\ v_i, v_j = \infty \\ p\ v_i, p^r\ v_i, v_j \text{ , если } m\ v_i, v_j > m^r\ v_i, v_j \text{ и } p\ v_i, p^r\ v_i, v_j \neq \infty \end{matrix} \quad (1.6)$$

где $v_i$ – первая вершина с координатами $(X_k, Y_l), k = 1 \dots N, l = 1 \dots N$,

$v_j$ – вторая вершина с координатами $(X_q, Y_p), q = 1 \dots N, p = 1 \dots N$,

$p^r\ v_i, v_j$ – элементы матрицы последователей $P$ графа $G$. Найденные пути между вершинами графа $G$ будут гарантированно кратчайшими.

Значения матриц $M$ и $P$ задаются следующим образом:

$$m\ v_i, v_j = u\ v_i, v_j \quad (1.7)$$

$$p\ v_i, v_j = \begin{matrix} v_i, \text{если } u\ v_i, v_j \neq \infty \\ \emptyset, \text{если } u\ v_i, v_j = \infty \end{matrix} \quad (1.8)$$

Исходя из описанного выше, сформируем требования к разрабатываемому программному модулю:

- разрабатываемый программный модуль должен строить N графов минимальной длины;
- разрабатываемый программный модуль должен строить оптимальный сетевой граф;
- разрабатываемый программный модуль должен быть оснащен юнит-тестированием.

Определив требования к разрабатываемому программному модулю и описав математическое представление решаемой задачи, перейдем к сравнению существующий аналогов.

## 1.4 Реализация виртуализации сетевых функций различными компаниями

Реализацией виртуальных функций, помимо Неткрэкера, занимаются многие компании, так как виртуализация сетевых функций рассматривается операторами связи, как самое важное преобразование, которое сетевая индустрия предприняла в своей истории. NFV обещает предоставить операторам возможность увеличить доходы от услуг, автоматизировать многие, если не все, сетевые функции, сократить расходы на инфраструктуру, необходимую для поддержки новых услуг, и обеспечить масштаб, необходимый для удовлетворения растущих потребностей в пропускной способности трафика [13].

Снег Дэвид, главный аналитик американской IT-компании Hewlett Packard Enterprise, в своей статье от 30 июня 2016 года сравнил существующие реализации MANO. Опираясь на его исследования, проведем сравнение аналогов [13].

Amdocs - американская компания по производству программного обеспечения и линейки одноимённых продуктов. Сетевой администратор облачной службы Amdocs (Network Cloud Service Orchestrator, далее NCSO) хорошо работает практически по всем критериям. Компания была одной из первых, кто подчеркнул необходимость непрерывной работы в режиме реального времени и обеспечения услуг на основе NFV, используя свою технологию «Sensei». NCSO также предлагает тесную интеграцию с IT-активами Amdocs, особенно со стороны информации (данных). Компания добилась успеха во многих направлениях VNF. Тем не менее, представляя себя как разработчика VNF, так и оператора с инструментами для разработки систем, предложение Amdocs предлагается более целостным, чем другие.

Компания не делит свое решение на составляющие, клиент не может купить один модуль, если остальные ему не нужны. Так же продукт компании Amdocs не совместим со сторонними решениями [13][7].

Cisco - американская транснациональная компания, разрабатывающая и продающая сетевое оборудование, предназначенное в основном для крупных организаций и телекоммуникационных предприятий. В марте 2015 года стало известно, что произошло слияние с Tail-f Systems. В результате, Cisco получил VNFM Elastic Services Controller (ESC), которые представляют основные EOSI-совместимые активы MANO, являющиеся частью более широкого предложения NFV - Evolved Services Platform (ESP). В рамках своего предложения MANO, Cisco также добавила свою систему виртуальной топологии (Virtual Topology System, далее VTS) для расширения возможностей управления SDN и оснастила ее безопасностью и лицензированием в большей степени, чем у большинства других предложений. Однако, когда дело доходит до развертывания MANO, на разных уровнях оркестровки, компания не может раскрывать какую-либо количественную информацию, хотя указывает на работу с высококлассными операторами в США и Европе. Cisco должен обеспечить большую видимость своего предложения MANO в режиме реального времени, поддерживая сторонние VNF, чтобы подтвердить и продемонстрировать конкурентные преимущества своего решения [6][8][9].

Huawei - одна из крупнейших китайских компаний в сфере телекоммуникаций, основанная Жэнем Чжэнфэем в 1987 году. Предложение MANO от Huawei только недавно стало заметным на рынке, как отдельный элемент своего продукта Infrastructure Enabling System (далее IES) под маркой «CloudOpera». CloudOpera ICTOrchestrator, наряду с ICT-Assurance, виртуализирует и объединяет в облако SDN / NFV, устаревшие сетевые ресурсы и интерфейсы NFVO от Huawei. Huawei тщательно следит за возможностями своих заказчиков, широко участвует в испытаниях и развертываниях NFV, является одним из немногих поставщиков, который показывает, что его предложение находится в прямой эксплуатации. Очевидно,

что Huawei вложила много в уровень VIM (OpenStack), но его VNFM и NFVO еще не обеспечены какими-либо отличительными функциями. Компании следует рассмотреть вопрос о внедрении лицензирования VNF в свое предложение, которое в настоящее время рассматривается как выходящее за рамки NFV MANO [6][10][1].

Неткрэкер - дочерняя компания корпорации NEC, специализирующаяся на создании, внедрении и сопровождении систем эксплуатационной поддержки, систем поддержки бизнеса, а также SDN/NFV-решений для операторов связи, крупных предприятий и государственных учреждений. В то время как сложность IT-решений компании и последние предложения по внедрению SDN / NFV, платформа Agile Virtualization Platform (AVP) очень высока, вряд ли можно сомневаться в ее полноте в отношении платформы, способной поддерживать практически все аспекты. NFVO и VNFM от Неткрэкер, являются одними из наиболее функционально сложных, особенно в области безопасности и лицензирования, но эксплуатация и установка большого числа сторонних VNF, их согласование с уровнем VNFM обеспечивает лидирующее положение решения компании. Неткрэкер также успешно перенес эти возможности в реальные развертывания MANO на уровне NFVO. Тем не менее, в то время как платформа Неткрэкер предлагает очень обширные услуги, она может быть слишком тяжеловесной и монолитной для небольших операторов [6][11].

Oracle - американская транснациональная корпорация, второй по величине доходов производитель программного обеспечения (после Microsoft), крупнейший производитель программного обеспечения для организаций, крупный поставщик серверного оборудования. Сетевой сервис Oracle Orchestrator (OC NSO) обеспечивает все ожидаемые функции, будучи окруженным более широким портфелем предложений NFV от Oracle. NSO демонстрирует устойчивость как VNF-платформа с несколькими вендорами, интегрируя все большее количество сторонних партнеров VNF и способность организовать сложные сетевые сервисы. Oracle также демонстрирует четкое

различие между функциями взаимодействия с клиентами и сетью, демонстрирует практические инновации в области управления объединенными (PNF / VNF) сетевыми функциями в своем Oracle Application Application Orchestrator. Несмотря на то, что успехи развертывания на уровнях ни VNFM, и / или NFVO конфиденциальны, официальные лица указывают на увеличение спроса для NSO [6][12][13].

Из статьи Снега Дэвида [6] видно, что конкурентная среда четко отображает продолжающееся соперничество между сетевыми поставщиками на владение пространством NFV MANO. Ни один из поставщиков не открывает свой программный код, поэтому ведется параллельная разработка и сравнивать можно только результаты работы конечного продукта.

Сравнение решений разных компаний было проведено для того, чтобы показать, что виртуализация – это многообещающее направление, развивающееся по всему миру.

## Вывод по главе 1

В первой главе выпускной квалификационной работы была проанализирована существующая модель построения сетевых графов, были найдены проблемы данного процесса и их решение в виде его автоматизации. Рассмотрены алгоритмы построение оптимальных путей в графе, описана математическая модель представления сетевого графа, а также определены требования к разрабатываемому программному модулю. Рассмотрены существующие реализации MANO, проведено их сравнение.

# Глава 2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПОСТРОЕНИЯ СЕТЕВЫХ ГРАФОВ ПОСЛЕДОВАТЕЛЬНОСТИ ВИРТУАЛЬНЫХ ФУНКЦИЙ

## 2.1 Проектирование разрабатываемого программного модуля для построения сетевых графов последовательности виртуальных функций

### 2.1.1 Обоснование архитектуры разрабатываемого программного модуля построения сетевых графов

Определив недостатки алгоритма построения сетевых графов, был сделан вывод, что разработку программного модуля нужно строить согласно водопадной модели. Водопадная модель жизненного цикла предусматривает выполнение всех этапов проектирования в строго установленном порядке. Переход работы к следующему этапу возможен только в том случае, если полностью завершен предыдущий.

На начальном этапе определимся с архитектурой разрабатываемого программного модуля. В качестве шаблона проектирования был выбран паттерн «Модель-Представление-Контроллер» (Model-View-Controller pattern, рисунок 12), так как он позволяет изменять свои компоненты независимо друг от друга.

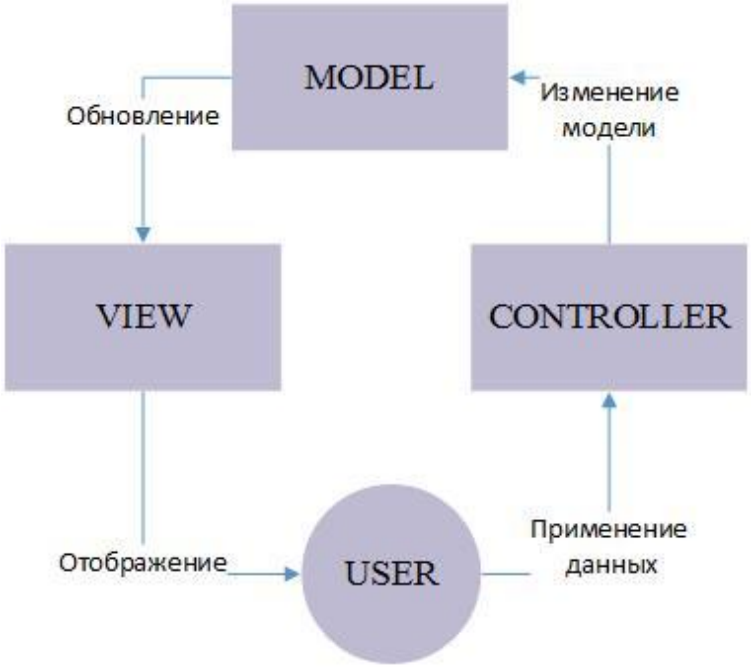


Рисунок 12 - Архитектура программного модуля

Элементы паттерна:

- Model – модель, предоставляющая данные и реагирующая на команды контроллера, изменяя свое состояние;
- View – представление, которое отвечает за отображение данных модели пользователю, реагируя на изменения модели;
- Controller - контроллер интерпретирует действия пользователя, оповещая модель о необходимости изменений;
- User – пользователь, работающий с программным модулем.

В роли User будет выступать NS-оператор. Создав и настроив VNFs, он передаст данные о них контроллеру, тот в свою очередь изменит модель, в ней произойдет построение оптимального графа, данные о нем буду отправлены на View, построенный сетевой граф отобразится пользователю.

После описания архитектуры разрабатываемого программного модуля определимся, как будет развернут программный модуль на физических устройствах. Для этого рассмотрим диаграмму развертывания, которая представлена на рисунке 13.

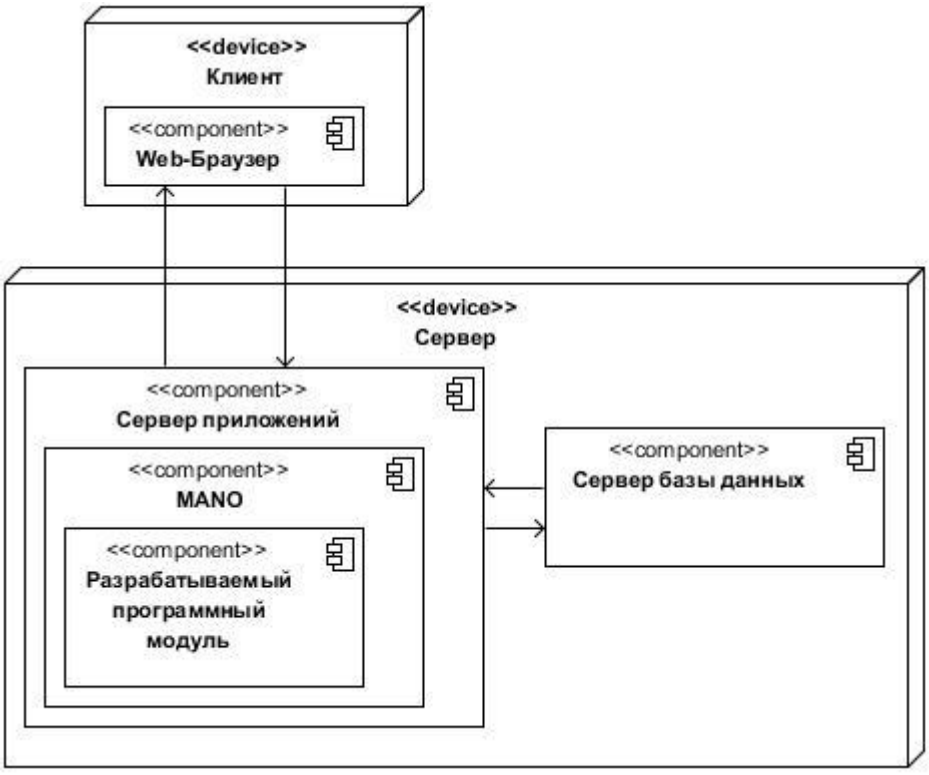


Рисунок 13 - Диаграмма развертывания разрабатываемого программного модуля

На диаграмме видно, что разрабатываемый программный модуль будет встроен в продукт компании Неткрэкер – MANO, который, в свою очередь, будет взаимодействовать с сервером приложений и сервером базы данных.

Описав диаграмму развертывания программного модуля, сформулируем требования к разрабатываемому программному модулю на основе классификации требований FURPS+ (таблица 2).

Таблица 2 - Требования к разрабатываемому программному модулю

| ID | Требование | Статус | Полезность | Риск | Стабильность |
|---|---|---|---|---|---|
| **Functionality — функциональные требования** | | | | | |
| 1 | Строить оптимальный сетевой граф | Одобрено | Критичное | Средней | Средняя |
| **Usability — требования к удобству использования** | | | | | |
| 2 | Размер основного шрифта должен быть 14px | Одобрено | Важное | Низкий | Низкая |
| 3 | Построенный граф, текст на нем и фон должны быть достаточно контрастными | Одобрено | Важное | Низкий | Низкая |
| 4 | Разрешение экрана не менее 1145x768 | Одобрено | Важное | Низкий | Низкая |
| **Reliability — требования к надежности** | | | | | |
| 5 | Доступ к системе в режиме 24/7/365 | Одобрено | Критичное | Низкий | Низкая |
| **Performance — требования к производительности** | | | | | |
| 6 | Время реакции системы не более двух секунд | Предложенные | Важное | Средний | Низкая |

Продолжение таблицы 2

| ID | Требование | Статус | Полезность | Риск | Стабильность |
|---|---|---|---|---|---|
| **Supportability — требования к поддержке** | | | | | |
| 7 | Время устранения неисправностей не более 30 минут | Предложенные | Важное | Средний | Низкая |
| **+ ограничения** | | | | | |
| **Ограничения реализации** | | | | | |
| 8 | Реализация логики должна быть написана на языке программирования Java версии 8.0 | Одобрено | Критичное | Низкий | Высокая |
| **Ограничения проектирования** | | | | | |
| 9 | Программный модуль должен быть совместим с программным продуктом компании «Неткрэкер» | Одобрено | Критичное | Низкий | Высокая |

Из таблицы 2 видно, что реализация программного модуля должна выполняться на Java-платформе, с использованием языка Java версии 8.0, а также должна поддерживать совместимость с программным продуктом компании «Неткрэкер». Для этого требуется использовать универсальный фреймворк Spring и веб-сервер WildFly. В следующем пункте приведем обоснование выбора компанией данных технологий.

### 2.1.2 Обоснование выбора средств реализации разрабатываемого программного модуля для построения сетевых графов

Заказчиком работы, «Неткрэкером», было обозначено, что программный модуль должен быть написан на языке программирования Java и оптимизирован для работы на веб-сервере WildFly. Язык программирования

Java выбран «Неткрэкером» потому, что он обладает рядом преимуществ, например:

- приложения Java транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, с помощью виртуальной Java-машины [1];
- язык программирования Java является объектно-ориентированным [1];
- Java разработан компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Oracle документирует все изменения, включенные в новые версии языка, имеется большое количество печатных изданий, в которых описаны нюансы работы на языке Java [1];
- Java – «живой» язык программирования, который не стоит на месте, регулярно выходят новые версии с исправленными ошибками и новыми функциональными возможностями [1].

Как видно из преимуществ языка программирования Java, он хорошо подходит для решения задач, поставленных в ВКР.

Дадим краткую характеристику фреймворку Spring. Это самая популярная среда разработки для Java приложений. Миллионы разработчиков во всем мире используют Spring для создания высокопроизводительного, легко тестируемого кода. Spring не занимает много места - базовая версия имеет размер около 2 МБ.

Преимущества использования Spring Framework:

- Spring позволяет разработчикам разрабатывать приложения корпоративного класса с использованием POJO. Преимущество использования POJO заключается в том, что разработчик может отказаться от продукта EJB, но может использовать различные контейнеры сервлетов, например, Tomcat или другие [2];
- Spring организован модульно. Несмотря на то, что он имеет большое количество пакетов и классов, разработчику нужно следить только за теми, которые он использует, игнорируя остальные [2];

- тестировать приложения, написанные с помощью Spring, просто, потому что в эту среду интегрируется зависящий от среды код [2];
- веб-структура Spring - это хорошо разработанная веб-структура MVC, которая представляет собой отличную альтернативу веб-фреймворкам, таким как Struts и другие [2];
- Spring предоставляет последовательный интерфейс управления транзакциями, который может масштабироваться до локальной транзакции (например, с помощью одной базы данных) и масштабироваться до глобальных транзакций [2].

Основные функции Spring могут быть использованы при разработке любых приложений Java, существуют расширения для создания веб-приложений поверх платформы Java EE. Spring ставит перед собой задачу облегчить использование J2EE-разработки и продвигать хорошие методы программирования за счет включения POJO-модели программирования [2]. Именно поэтому компания Неткрэкер отдала свое предпочтение этому фреймворку.

Сервер приложений WildFly так же выбран компанией Неткрэкер потому, что он занимает лидирующую позицию среди аналогов. WildFly обладает следующими преимуществами:

- высокая скорость выполнения: быстрый запуск обеспечен тем, что сервисы запускаются одновременно, чтобы исключить ненужные ожидания и задействовать многоядерные процессоры [3];
- максимальная производительность и масштабируемость веб-сайтов. Связность, отзывчивость и масштабируемость имеют первостепенное значение для современных веб-приложений. Для удовлетворения этих требований был разработан новый гибкий, высокопроизводительный веб-сервер под названием Undertow, и он является неотъемлемой частью WildFly 8. Undertow имеет возможность масштабирования до более миллиона подключений, исследование сторонних производителей показали, что он лучше конкурентов в пропускной способности [3];

- WildFly использует общие кэшированные индексированные метаданные для дублирования полных анализов, что уменьшает количество используемой памяти и оттока объектов. Использование модульной загрузки классов предотвращает дублирование классов, это не только снижает нагрузку на память, но и помогает минимизировать паузы сборщика мусора [3];
- конфигурационный файл организован подсистемами. Подсистемы используют интеллектуальные значения по умолчанию, но их можно настроить так, чтобы они наилучшим образом соответствовали потребностям пользователя. При работе в режиме домена, конфигурация для всех серверов, участвующих в домене, хорошо организована в одном файле [3];
- изменения в конфигурации не ограничиваются редактированием файлов. Все возможности управления раскрываются унифицированным образом во многих формах доступа. К ним относятся веб-консоль администрирования, собственный Java API, шлюз JMX. Эти параметры позволяют настраивать автоматизацию с использованием инструментов и языков, которые наилучшим образом соответствуют потребностям пользователя [3];
- WildFly поддерживает новейшие стандарты веб-разработки. Поддержка веб-сокетов позволяет приложениям использовать оптимизированные настраиваемые протоколы и полнодуплексную связь с базовой инфраструктурой. Это особенно полезно при общении с мобильными устройствами [3];
- отказоустойчивость, кластеризация, репликация сеансов и эффективный веб-прокси-сервер вкладываются в качестве функций базового уровня WildFly [3].

Из перечисленных преимуществ видно, что WildFly один из лучших серверов приложений и его выбор оправдан.

Определившись со средствами разработки, перейдем к реализации программного модуля, отвечающего за оптимальное построение сетевого графа последовательности виртуальных функций.

## 2.2 Реализация программного модуля для построения оптимального сетевого графа последовательности виртуальных функций

Организация программы в виде совокупности небольших независимых блоков, которые называют модулями, структура и поведение которых подчиняются определённым правилам, называют модульным программированием. Этот подход позволяет упростить тестирование разработанного программного кода, обнаружение и исправление ошибок. Такой подход позволяет отделить аппаратно-зависимые подзадачи от других подзадач, что улучшает мобильность создаваемых программ.

Под термином «модуль» понимают функционально законченный фрагмент программы, для которого указывается реализуемая им функциональности и связь с другими модулями. Сервисы, классы, библиотеки функций, структуры данных могут играть роль модуля.

Принцип модульного программирования обладает следующими достоинствами:

- упрощает задачи проектирования системы;
- упрощает распределение процесса разработки между группами разработчиков;
- наличие возможности изменения отдельно взятого модуля, без необходимости изменения всей системы;
- уменьшается время перекомпиляции при изменениях программного кода, так как каждый модуль компилируется отдельно;
- возможность замены отдельного компонента конечного продукта без необходимости изменения настроек;

Для реализации алгоритма решения задачи, поставленной в первой главе, составим блок-схему одного из модулей - заполнения матрицы расстояний (рисунок 14).

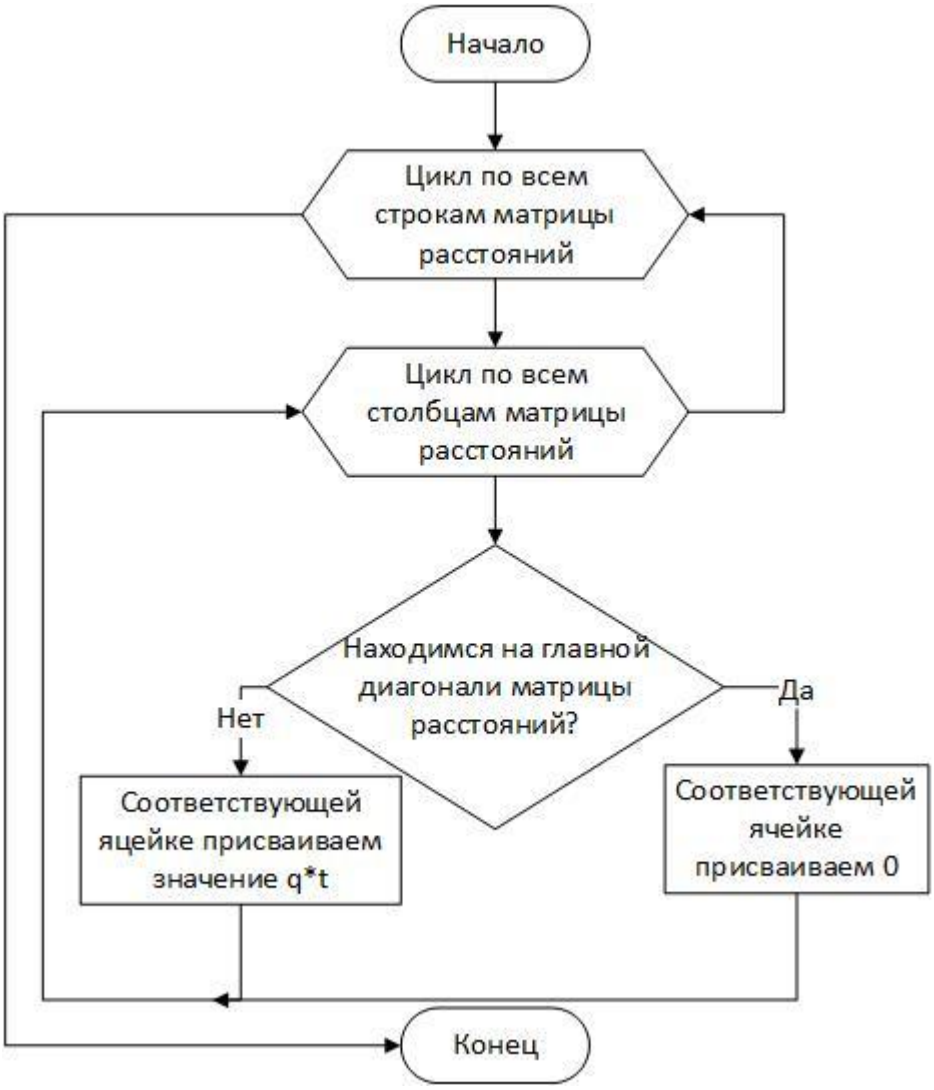


Рисунок 14 - Алгоритм заполнения матрицы расстояний

Матрица расстояний, используемая в алгоритме, представляет собой двумерный массив. Главная диагональ матрицы расстояний равна нулю, поэтому при выполнении условия, элементу массива присваивается 0. Если же условие не выполняется, то значение элемента вычисляется по формуле (1.2). На блок-схеме q = {0,1} – булева функция, значение которой присваивается в зависимости от наличия отклика сетевой функции на сигнал, а t – время выполнения сетевой функцией тестового запроса.

Заполнив матрицу расстояний, можем приступать к реализации метода построения графа. На рисунке 15 представлен алгоритм выполнения функции построения графа, который рекурсивно вызывается для всех вершин, имеющихся на графике.


Рисунок 15 - Алгоритм построения графа

После рекурсивного выполнения метода построения графа для всех вершин, можем выбрать оптимальный граф из построенных и применить его к NS и созданным VNFs (листинг программного кода представлен в приложении А).

## 2.3 Описание примера работы программного модуля построения оптимального сетевого графа

Рассмотрим пример построения сетевого графа, с помощью разработанного программного модуля, для трех виртуальных сетевых функций. На рисунке 16 представлен пример входных данных, где

- VNF – созданные виртуальные сетевые функции;

- CP – точки подключения, созданные NS-оператором:
  - LCP – точка подключения для приема трафика;
  - RCP – точка подключения для отдачи трафика;
  - MCP – точка подключения для настройки виртуальной сетевой функции;
- NS – набор виртуальных сетевых функций.

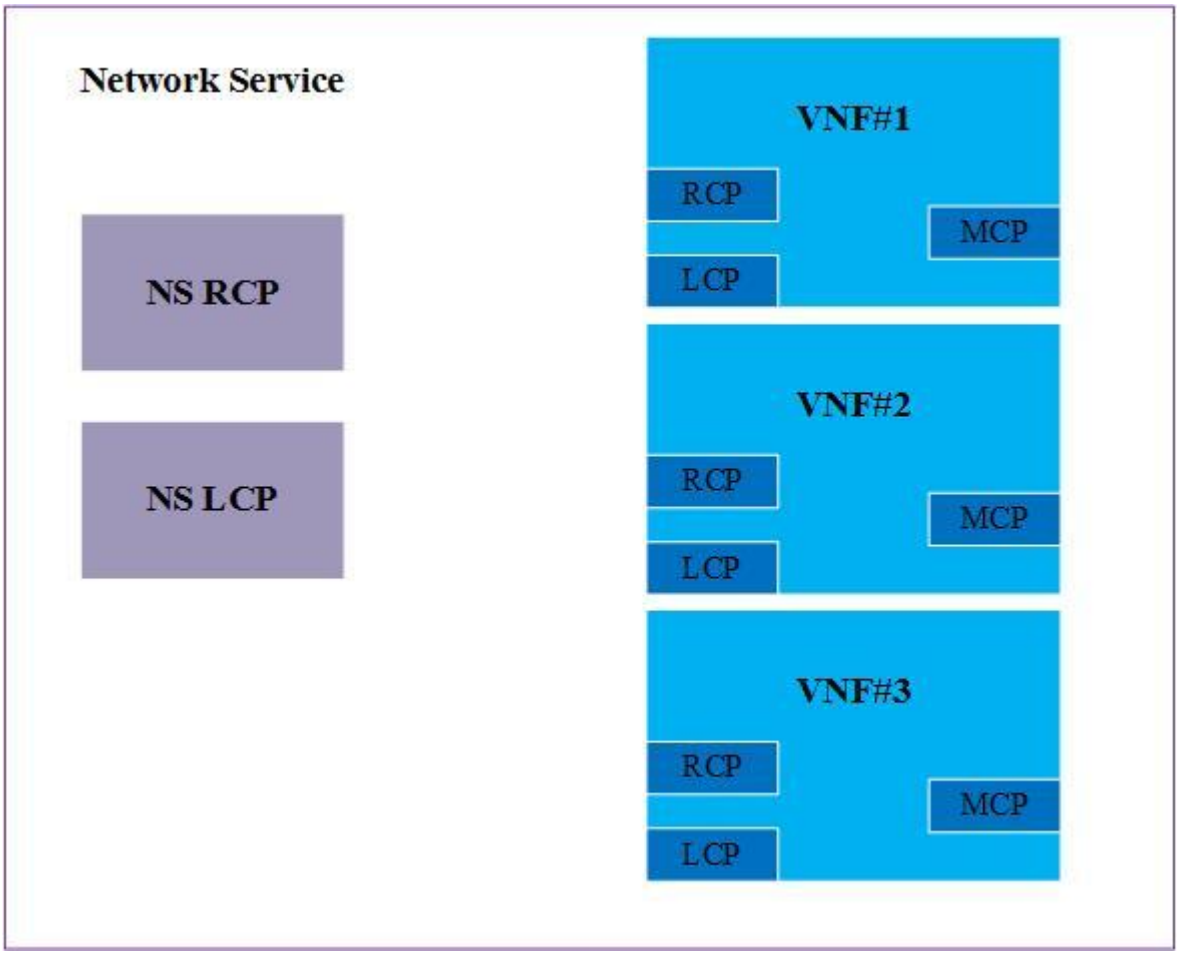


Рисунок 16 - Входные данные

После применения разработанного программного модуля, получаем построенный сетевой график пересылки трафика (рисунок 17).

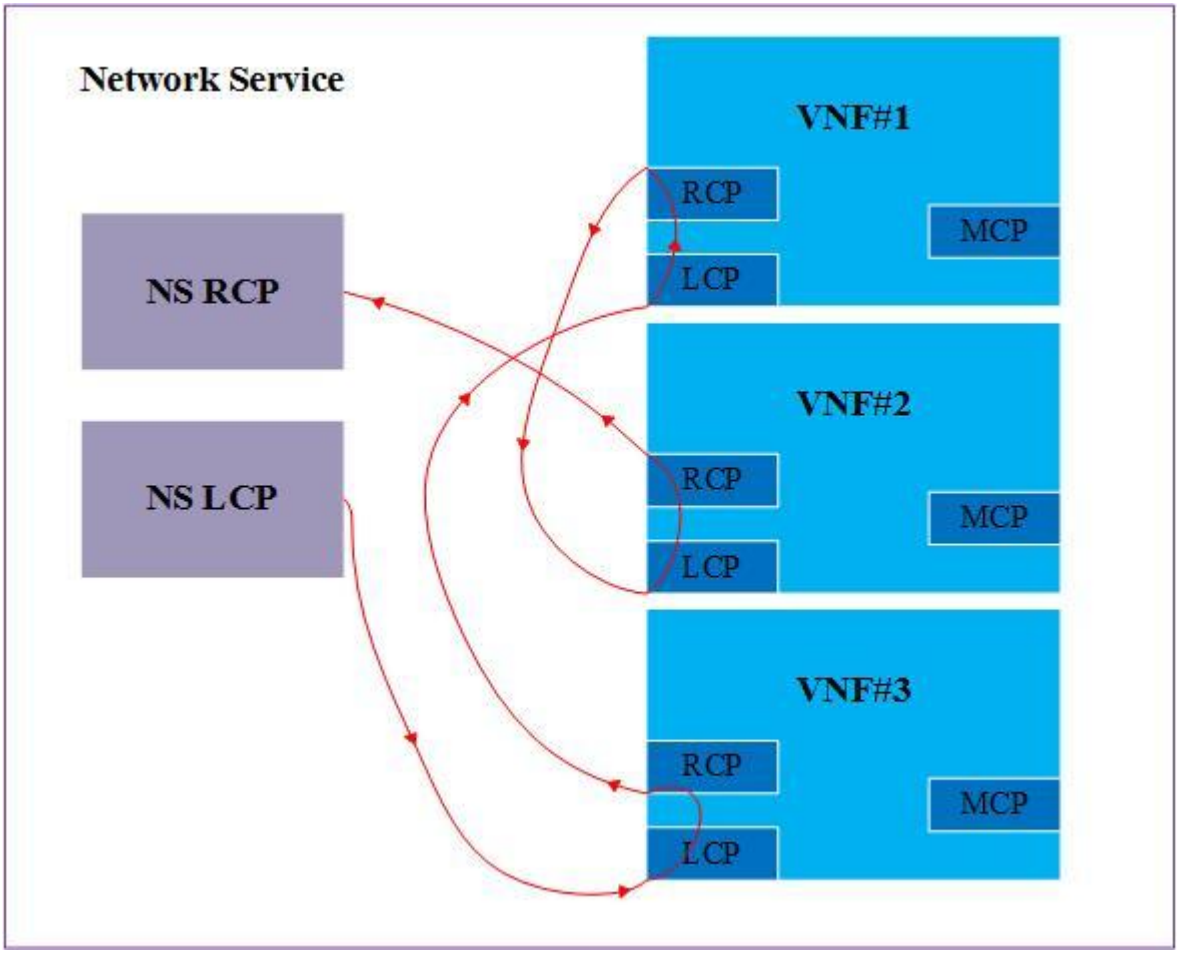


Рисунок 17 - Результат работы программного модуля

На рисунке 17 видно, что построен сетевой граф, проходящий через узлы в следующем порядке: VNF#3, VNF#1, VNF#2. Проверим, является ли он оптимальным, построив матрицу расстояний (таблица 3).

Таблица 3 - Матрица расстояний

| | **VNF#1** | **VNF#2** | **VNF#3** |
|---|---|---|---|
| **VNF#1** | 0 | 2 | 1 |
| **VNF#2** | 2 | 0 | 3 |
| **VNF#3** | 1 | 3 | 0 |

Данный набор VNFs является тестовым, создан в качестве примера, поэтому значения расстояний имеют вид, удобный для ручного подсчета. Значения матрицы были получены с помощью метода заполнения матрицы

расстояний, описанного выше. Построим все возможные пути графа и посчитаем значения $s_i$, используя формулу (1.2):

- путь VNF#1, VNF#2, VNF#3: $s_1 = 5$;
- путь VNF#1, VNF#3, VNF#2: $s_2 = 4$;
- путь VNF#2, VNF#1, VNF#3: $s_3 = 3$;
- путь VNF#2, VNF#3, VNF#1: $s_4 = 4$;
- путь VNF#3, VNF#1, VNF#2: $s_5 = 3$;
- путь VNF#3, VNF#2, VNF#1: $s_6 = 5$.

Из найденных значений видим, что пути VNF#2, VNF#1, VNF#3 и VNF#3, VNF#1, VNF#2 имеют равные значения $s_i$ и являются минимальными. Разработанный программный модуль построил граф по пути VNF#3, VNF#1, VNF#2. Такой выбор считается допустимым при наличии двух путей равной длины. Для данного набора сетевых функций прохождение трафика в указанном порядке занимает наименьшее время.

Описав разработку программного модуля для построения оптимального сетевого графа и представив результаты работы, перейдем к его тестированию.

## 2.4 Тестирование разработанного программного модуля для построения оптимального сетевого графа

Тестирование программного обеспечения — процесс испытания ПО с целью получения информации о качестве продукта.

Задача тестирования программных модулей состоит проверке следующих параметров:

- корректного взаимодействия между функциями;
- соответствие данных на входе и выходе взаимодействующих программных модулей и групп программ.

Тестирование заключается в выполнении приложения на некотором множестве исходных данных. После выполнения, полученные результаты сравниваются с заранее известными (эталонными), чтобы установить

соответствие различных свойств и характеристик приложения ожидаемым свойствам.

Тестирование обычно производится на протяжении всей разработки и сопровождения на разных уровнях. При конструировании используются две формы тестирования, проводимого программистами, непосредственно создающими исходный код:

- модульное тестирование (unit testing) - уровень тестирования, позволяющий проверить функционирование отдельно взятого элемента системы;
- интеграционное тестирование (integration testing) - уровень тестирования, являющийся процессом проверки взаимодействия между программными модулями.

Рассмотрим пример тестирования разработанного программного модуля при помощи модульного тестирования.

Для создания тестового класса нужно унаследовать его от класса TestCase и переопределить методы setUp() и tearDown(). При создании тестового метода необходимо пометить его аннотацией @Test. При запуске создастся экземпляр тестового класса, после, методы вызовутся в следующем порядке:

- setUp();
- методы, помеченные аннотацией @Test;
- tearDown().

В случае, если хотя бы один из методов выбросит исключение, тест будет считаться провальным, компиляция завершится в аварийном режиме с указанием на провалившийся тестовый метод.

Тестовые методы чаще всего состоят из некоторого программного кода и сравнений полученных значений с ожидаемыми значениями при помощи класса Assert или ключевого слова assert.

Например, запустив тестовый метод для графа с заранее заданными матрицами расстояний, можем выяснить работоспособность разработанного программного модуля (рисунок 18).

```
@Test
public void testBuildOptimalGraph()
{
    Graph actual = new Graph();
    assertFalse(actual.buildOptimalGraph(wrongDistanse));
}
```

Рисунок 18 - Листинг тестового метода testBuildOptimalGraph

В представленном тесте проверяется корректность работы метода buildOptimalGraph. Посылая на вход заведомо некорректные данные, метод должен вернуть false. Запустив тот же метод с заведомо корректными данными, на выходе ожидаем true (рисунок 19).

```
@Test
public void testBuildOptimalGraph()
{
    Graph actual = new Graph();
    assertTrue(actual.buildOptimalGraph(optimalDistanse));
}
```

Рисунок 19 - Листинг тестового метода testBuildOptimalGraph

Следующий тест, представленный на рисунке 20, проверяет, действительно ли построенный граф является оптимальным.

```
@Test
public void testValidateOptimalGraph()
{
    Graph graph = new Graph();
    graph = graph.buildOptimalGraph(distances);
    assertEquals(optimalGraph, graph);

}
```

Рисунок 20 - Листинг тестового метода testValidateOptimalGraph

На вход методу buildOptimalGraph подается матрица расстояний distances, в которой содержатся заранее заданные значения. Сравнивая построенный граф

graph, ожидаем совпадения с заранее известным оптимальным графом optimalGraph для матрицы расстояний distances.

Оценив результаты тестирования разработанного программного модуля, можно сделать вывод, что его работа корректна. Программный модуль проверяет корректность входных данных и строит оптимальный сетевой граф последовательности виртуальных функций. Оптимальность построенного сетевого графа последовательности виртуальных функций была доказана, из этого можно сделать вывод, что разработанный алгоритм отвечает поставленным функциональным требованиям.

## Вывод по главе 2

Во второй главе была описана архитектура разрабатываемого программного модуля, диаграмма развертывания, описывающая физические носители, на которых будут располагаться модульные компоненты системы, сформулированы требования к программному модулю, обоснованы выбранные средства разработки, составлены и описаны блок-схемы программного модуля, рассмотрены варианты тестирования программного модуля.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была проанализирована литература по проблемам построения сетевого графа. После анализа научно-методической литературы были определены основные алгоритмы построения кратчайшего пути в сетевом графе.

При выборе алгоритма построения кратчайшего пути в графе, были изучены два вида задач MDSP и их решения. Данное исследование позволило выбрать алгоритм построения оптимального сетевого графа.

Проанализировав существующий алгоритм построение сетевого графа, были выявлены минусы этого процесса, а также найдена последовательность действий NS-оператора, требующая автоматизации. На графическом языке объектного моделирования UML была смоделирована диаграмма деятельности NS-оператора, что позволило определить основные функциональные требования к разрабатываемому программному модулю.

Выделив модуль, требующий автоматизации, был представлен новый алгоритм построения сетевого графа, избавляющий оператора от монотонного выполнения функций прокладки путей прохождения трафика в сетевом графе.

Выполнено проектирование программного модуля, необходимое для дальнейшей реализации. В проектирование вошли следующие этапы:

- определение архитектуры разрабатываемого программного модуля, что помогло понять, какие компоненты он будет включать;
- описание физических носителей, на которых будут располагаться модульные компоненты системы;
- определение требований к программному модулю, на основе классификации требований FURBS+.

Результаты проектирования программного модуля помогли определить компоненты, которые он должен включать для реализации требований. Чтобы уменьшить количество ошибок при реализации и для увеличения ее скорости, были представлены алгоритмы работы методов заполнения матрицы расстояний и построения оптимального сетевого графа.

В результате выполнения выпускной квалификационной работы, был разработан программный модуль для управления последовательностью виртуальных функций программно-определяемой сети SDN. Данный модуль позволит сэкономить время NS-операторов, необходимое для построения сетевого графа. Главным достоинством программного модуля является то, что он строит оптимальный сетевой граф последовательности виртуальных функций.

После выполнения выпускной квалификационной работы сделан вывод, что использование программно-определяемых сетей SDN является оптимальным решением для построения виртуальных сетей. Виртуализация сетевых функций необходима современным поставщикам услуг связи из-за увеличивающегося в геометрической прогрессии количества информации. Виртуальные функции и программно-определяемые сети – это интересное, многообещающее направление, которое сможет обеспечить бесперебойную работу поставщиков услуг связи, а также снизить стоимость построения сетей.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

*Научная и методическая литература*

1. Божко В. П. Информационные технологии в статистике: учеб.-практ. пособие / В. П. Божко. - Москва : Евразийский открытый институт, 2010. - 167 с.

2. Демидов Д. В. Использование реляционной теории при оптимальном проектировании интегральных схем: науч. работа / Д. В. Демидов – Санкт-Петербург : ИТМО, 2015. – 79 с.

3. Информационные технологии в профессиональной деятельности: учеб. пособие / авт.-сост. А. А. Широких. - Пермь : Пермский гос. гуманит.-пед. ун-т, 2014. - 61 с.

4. Клочко И. А. Информационные технологии в профессиональной деятельности: учеб. пособие / И. А. Клочко. - Саратов : Вузовское образование, 2014. - 236 с.

5. Мишин А. В. Информационные технологии в профессиональной деятельности : учебное пособие / А. В. Мишин, Л. Е. Мистров, Д. В. Картавцев. - Москва : Российская академия правосудия, 2011. - 311 с.

6. Силич В. А. Реинжиниринг бизнес-процессов: учеб. пособие / В. А. Силич, М. П. Силич. - Томск : ТУСУР, 2014. - 199 с.

7. Тимеряев Т. В. Методы и алгоритмы управления маршрутизацией в транспортных сетях на основе оперативной обработки информации в разреженных графах: науч. работа / Т. В. Тимеряев – Уфа : ФГБОУ ВПО, 2015. – 203 с.

8. Юдин К. А. Автоматизация проектирования с применением Autodesk Inventor 2012: учеб. пособие / К. А. Юдин ; Белгородский гос. технол. ун-т им. В. Г. Шухова. - Белгород : БГТУ : ЭБС АСВ, 2013. - 128 с.

*Электронные ресурсы*

9. NEC and NetCracker Launch New Business Brand and Optimize NEC’s Network Expertise and NetCracker’s IT Leadership [Электронный ресурс]. – Режим доступа: http://www.nec.com/en/press/201502/global_20150224_01.html.

10. NEC Announces Agreement to Acquire Convergys' Global Information Management (IM) Business, a Leader in Business Support Systems (BSS) [Электронный ресурс]. – Режим доступа: http://www.nec.co.jp/press/en/1203/2202.html.

11. NEC Completes Acquisition of Convergys Corporation's Information Management (IM) Business [Электронный ресурс]. – Режим доступа: http://www.nec.com/en/press/201205/global_20120517_02.html.

12. NetCracker Announces Agreement to Acquire Subex's Activation Business [Электронный ресурс]. – Режим доступа: http://www.businesswire.com/news/home.

13. Supporting the Profitable Operator of the Future [Электронный ресурс]. – Режим доступа: http://www.mobileeurope.co.uk/.

14. Втюрин В.А. Компьютерные технологии в области автоматизации и управления. [Электронный ресурс] Учебное пособие по направлению 220700 " Автоматизация технологических процессов". – СПб: СПбГЛТУ. 2011. – 103 с. – Режим доступа: http://window.edu.ru/resource/063/77063.

15. Дюженкова Н.В., Молоткова Н.В., Радько О.Ю., Хазанова Д.Л., Уляхин Т.М. Технология и организация практической деятельности в сфере бизнес-информатики. Организация учебной и производственной практики [Электронный ресурс]: Учебное пособие. – Тамбов: Издательство ТГТУ, 2010. – 80 с. – Режим доступа: http://window.edu.ru/resource/101/73101.

16. Кафедра телекоммуникационных сетей и систем [Электронный ресурс]. – Режим доступа: http://iss.fizteh.ru/about.html.

17. Платунова С.М. Методы проектирования фрагментов компьютерной сети [Электронный ресурс]: Учебное пособие. – СПб.: НИУ ИТМО, 2012. – 51 с. – Режим доступа: http://window.edu.ru/resource/571/78571.

18. Трутнев Д.Р. Архитектуры информационных систем. Основы проектирования [Электронный ресурс]: Учебное пособие. – СПб.: НИУ ИТМО, 2012. – 66 с. – Режим доступа: http://window.edu.ru/resource/174/78174.

*Литература на иностранных языках*

19. Dr. Danny Coward Java EE 7. The Big Picture / McGraw-Hill – 2015 – 513 p.

20. Farrell J. Java Programming / Course Technology 2015 – 1026 p.

21. Hof P. van't, Kamiński M., Paulusma D., Szeider S., Thilikos D.M. On Graph Contractions and Induced Minors // Discrete Applied Mathematics. 2012. Vol. 160. P.

22. Oaks S. Java Performance: The Definitive Guide – O'Reilly, 2014.

23. Pilgrim, P. Digital Java EE 7 Web Application Development / P. Pilgrim — Packt Publishing, 2015. – 486 c.

24. Sam Newman, Building Microservices - O'Reilly Media, 2015. – 280 pages.

## ПРИЛОЖЕНИЕ А
## Листинг кода для определения кратчайшего пути с учетом длины ребра

```
public class Path {
    public int[] path;
}
public class Graph {
// Класс Arc представляет дугу, ведущую в узел end
    static class Arc {
        int end; // номер узла, в который входит эта дуга
        Arc next; // следующая дуга в списке
        int length; // длина дуги
        public Arc(int e, Arc n, int len) {
            end = e;
            next = n;
            length = len;
        }
    }
public int getDijkstraPath(int beg, int end, Path path) {
        if (beg == end) {
            path.path = new int[]{beg};
            return 0;
        }
// Выбираем начальную вершину в качестве исходной
        int selected = beg;
//     Вектор,     представляющий     множество     вершин,
рассматриваемых на следующем шаге
        Vector arrNext = new Vector();
        Set  setNotPassed  =  {new  Set(0,  vertexCount()  -
1)).inverse();
        int[] marks = new int[vertexCount()];
        int[] length = new int[vertexCount()];
// Инициализация массива меток
        for (int i = 0; i < marks.length; i++) {
            marks[1] = -1;
            length[1] = 0;
        }
        setNotPassed.remove(selected);
        arrNext.add(new Integer(selected));
// "Основной цикл обхода графа
        while (setNotPassed.card() > 0) {
// Поиск минимальной дуги
            Arc     minArc     =     new     Arc(O,     null,
Integer.MAX_VALUE);
```

```
                int minVertex = -1;
                for (int i = 0; i < arrNext.size(); i++) {
                    int       vertex       =       ((Integer)
arrNext.get(i)).intValue();
                    for (Arc a = graph[vertex]; a != null; a =
a.next) {
                        if     (setNotPassed.has(a.end)     &&
a.length < minArc.length) {
                            minArc = a;
                        }
                        minVertex = vertex;
                    }
                }
            }
            if (minArc.length == Integer.MAX_VALUE) {
    // Больше нет ни одной новой вершины
                break;
            }
            arrNext.add(new Integer(minArc.end));
            marks[minArc.end] = minVertex;
            length[minArc.end] = minArc.length;
            setNotPassed.remove(minArc.end);
            if (minArc.end == end) {
    // Конечная вершина достигнута
                break;
            }
        }
        if (marks[end] == -1) {
    // Путь в конечную вершину из начальной не найден
    path.path = new int[]{};
            return 0;
        }else {
    // Формирование результирующего массива
            int wholeLength = 0;
            path.path = new int[count];
            for (int i = end; marks[i] != -1; i = marks[i]) {
                path.path[--count] = i;
                wholeLength += length[i];
            }
            path.path[0] = beg;
            return wholeLength;
        }}}
```