МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК Заведующий кафедрой

д-р физ.-мат. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.И. Миков.

(подпись) (инициалы, фамилия)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**РАЗРАБОТКА СИСТЕМЫ МОДЕЛИРОВАНИЯ AD HOC СЕТЕЙ В**

**НЕВЫПУКЛЫХ ОБЛАСТЯХ**

Работу выполнил С. Ю. Чуприн

Факультет компьютерных технологий и прикладной математики Направление 02.03.02 – Фундаментальная информатика и информационные технологии

Научный руководитель, преп.\_\_\_\_\_\_\_\_ С.С. Ермоленко

Нормоконтролер Ю.С. Фисун

Краснодар 2016

РЕФЕРАТ

Выпускная квалификационная работа содержит 42 страницы, 15 рисунков, 6 источников, 1 приложение.

КЛИЕНТ-СЕРВЕР, AD-HOC СЕТИ, ANDROID, БД, МОБИЛЬНАЯ РАЗРАБОТКА, ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ OPENMP

Темой работы является разработка системы моделирования Ad-hoc сетей в невыпуклых областях

Объектом исследования данной работы является клиент-серверная архитектура ПО, модели Ad-hoc сетей.

Целью данной работы является построение клиент-серверной архитектуры для организации удаленных вычислений и статистического моделирования Ad-hoc сетей.

Для исследования и решения поставленной задачи использованы: для реализации клиента выбрана среда программирования Android Studio 2.2, для тестирования были выбраны эмулятор Genymotion и реальное устройство Nexus 5. Для реализации серверной части Sublime Text 2 и БД MySQL, язык

PHP, для разработки вычислительного ядра – программы, моделирующей Adhoc сети, была выбрана IDE Visual Studio, язык C++ и основные геометрические познания.

СОДЕРЖАНИЕ

Введение…………………………………………………………………. 3

1. Моделирование беспроводных Ad-hoc сетей………………………... 5
   1. Общие сведения о генерации сетей…………………………… 5
   2. Общие сведения об анализируемых характеристиках ……… 7
2. Способы и технологии параллельного программирования…………. 9
   1. Общие сведения о параллельном программировании……….. 9
   2. Краткий обзор используемой параллельной технологий…… 12
3. Архитектура клиент-сервер…………………………………………… 13
4. Программирование для мобильных платформ……………………… 17
   1. Общие сведения о мобильной разработке……………………. 17
   2. Архитектурный паттерн MVP……………………………........ 21
5. Состав программ и основные алгоритмы…………………………..... 24
   1. Программа моделирования Ad-hoc сетей…………………….. 24
      1. Генерация узлов…………………………………………….. 25
      2. Установка соединений между узлами сети………………. 26
   2. Распараллеливание вычислений ………………………......... 28
   3. Организация сервера………………………………………..... 29
      1. Алгоритм обработки запросов………………………........ 30
   4. Структура проекта клиентского Android-приложения …… 32 5.4.1 Логика работы клиентского приложения……………. 34
6. Демонстрация работы мобильного клиента……………………….. 35

Заключение……………………………………………………………… 42

Список использованных источников…………………………………. 43

Приложение А Состав программ ……………………………………….. 44

ВВЕДЕНИЕ

Компьютерные ad hoc сети– это беспроводные децентрализованные сети с *n* узлами, расположенными в некоторой ограниченной области *S*. Каждый узел характеризуется радиусом действия приемопередатчика сигналов *r*. Два узла могут обмениваться информацией, если расстояние между ними меньше *r*.

В настоящее время интенсивно развивается научное направление в области построения телекоммуникационных систем с динамической топологией сети. Подобные системы получили название MANET (Mobile аd hoc Networks). Дальнейшая тенденция развития MANET – это их полная интеграция в другие системы связи общего или специального использования.

Основное назначение подобных сетей – организация связи между подвижными объектами – людьми, автомобилями, железнодорожным и водным транспортом, военными объектами, помощь при бедствиях или в случае экстренных ситуаций и происшествий.

Ad-hoc сети будут полезны в тех случаях, когда развертывание полноценной стационарной сети затруднено, дорого или невозможно.

Преимуществабеспроводных мобильных сетей над стационарными заключаются в надежности, быстроте развертывания, отказоустойчивости. Так как нет фиксированной топологии, Ad-hoc сети можно разворачивать где и когда угодно.

Недостаткимобильных сетей тоже есть. Это такие факторы, как небольшой радиус действия из-за необходимости экономии заряда мобильных устройств, а также естественных и искусственных препятствий, имеющихся на территории развертывания.

В случае комплексных сетевых систем развёртывание и анализ могут быть крайне сложными.

Адекватное сравнение с чисто теоретических позиций затруднено тем, что на процесс передачи данных в Ad-hoc сетях оказывает влияние большое число различных факторов, многие из которых носят случайных характер и слабо поддаются строгому математическому анализу.

Мобильные приложения **–** самый быстрый и удобный способ получения информации для пользователя, учитывая, что смартфоны и планшеты есть почти у каждого человека в мире.

Клиент-серверныйподход при разработке программного обеспечения в настоящее время получил широкое распространение и используется повсеместно. Используя такой способ, все вычисления, требующие вычислительных ресурсов, переносятся на высокопроизводительные сервера, экономя ресурсы мобильных устройств. От клиента требуется лишь активное Интернет-соединение. Задача выполняется на сервере, а на клиенте отображается лишь результат ее выполнения. Таким образом, задача упрощается и для пользователя, и для разработчика.

Целью работы является построение клиент-серверной архитектуры для выполнения задач статистического моделирования – удаленного генерирования сетей и анализ их характеристик, таких как количество компонент связности сгенерированной сети, зависимость математических ожиданий от количества вершин в ней и построение графиков для проведения аналитических исследований.

1. Моделирование беспроводных Ad-hoc сетей

1.1 Общие сведения о генерации сетей

В качестве математической модели Ad-hoc сети можно использовать неориентированный геометрический граф, где вершинами являются узлы сети, а ребрами – соединения между узлами. Так как топология сети заранее неизвестна, поэтому она будет случайной.

То есть генерируется случайным граф с n вершинами в некоторой области.

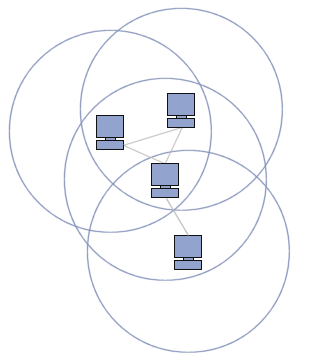


Рисунок 1 – Ad-hoc сеть

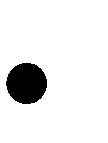
В дипломной работе будем считать, что сеть ограничена некоторой сферой с радиусом R, узлы сети также имеют вид сферы и радиус действия, равный r. Между двумя узлами устанавливается соединение, если расстояние между ними меньше r.

Распределение вероятностей координат узлов – трехмерное равномерное. Для случайного распределения узлов внутри общей сферы радиуса R используется сферическая система координат. Далее случайно сгенерированные сферические координаты узлов переводятся в декартовы координаты для удобства выполнения геометрических операций.

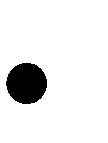
Сферическими координатами называют систему координат для отображения геометрических свойств фигуры в трёх измерениях посредством задания трёх координат, где r — кратчайшее расстояние до [начала координат,](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%87%D0%B0%D0%BB%D0%BE_%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82) а  и  — зенитный и азимутальный углы

соответственно.

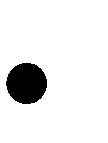
Три координаты (r,  , ), определены как:

 r ≥ 0 — расстояние от начала координат до заданной точки .

0 ≤ ≤ 180 º — угол между осью  и отрезком, соединяющим o начало координат и точку .



0 ≤ ≤ 360º — угол между осью  и проекцией отрезка, o соединяющего начало координат с точкой  на плоскость .



Важно заметить, что r < V, где V – радиус общей сферы.

Если заданы сферические координаты точки, то переход к декартовым осуществляется по формулам (1):



 *(1)*



Зенитный угол генерируется в пределах от 0 до 180°, азимутный в пределах от 0 до 360°. Преобразование сферических координат в декартовы происходит автоматически при создании нового узла и получении им сферических координат.

В программе используются следующие геометрические операции:

1. Нахождение расстояния от точки до точки в пространстве.
2. Нахождение модуля вектора.
3. Разность векторов.
4. Произведение векторов.
5. Нахождение расстояния от точки до прямой в пространстве.

1.2 Общие сведения об анализируемых характеристиках

Статистическое моделирование — исследование объектов познания на их статистических [моделях;](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) построение и изучение моделей реально существующих предметов, процессов или явлений с целью получения объяснений этих явлений, а также для предсказания явлений или показателей. В дипломной работе моделировать будем беспроводные мобильные Ad-hoc сети. Для моделирования будем генерировать сети при различные размерах общей сферы, размерах препятствия (случаи, когда препятствие занимает 0%, 10%, 20%, 50% от объема сферы) и при разных количествах генерируемых узлов сети. Для каждой новой полученной сети будем вычислять количество компонент связности, количество установленных соединений между узлами сети. Также необходимо определить зависимость математических ожиданий от количества вершин в графе.

Сбор данных будет происходить следующим образом: в цикле генерируются случайные графы, вычисляются характеристики. По окончании генерации полученные характеристики анализируются, создается статистика на основе этих аналитических данных. Результат будет представляться в виде графиков. Для большей наглядности результата эксперимента на одном графике будут выводиться значения без препятствия и с препятствиями различных радиусов.

Математическое ожидание – это среднее значение случайной величины. В нашем случае это будет среднее значение установленных соединений между узлами сети в зависимости от количества узлов.

*(2)*

Компонента связности графа — некоторое множество вершин [графа](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) такое, что для любых двух вершин из этого множества существует путь из одной в другую, и не существует пути из вершины этого множества в вершину не из этого множества. В сетях Ad-hoc связность важна по той причине, что от нее зависит целостность сети, а следовательно и общая работоспособность и надежность. Чем меньше компонент связности, тем больше узлов связаны между собой.

В программе для поиска компонент связности используется поиск в глубину.

Дисперсия случайной величины - мера разброса данной [случайной величины,](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0) то есть её отклонения от [математического ожидания.](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B6%D0%B8%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5)

Обозначается D[X].

Пусть Х — случайная величина, определённая на некотором [вероятностном пространстве.](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE) Тогда дисперсией называется формула на рисунке 9, где символ М обозначает [математическое ожидание.](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B6%D0%B8%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5)

*(3)*

1. Способы и средства параллельного программирования

2.1 Общие сведения о параллельном программировании

Параллельное программирование – способ написания компьютерных программ, при котором программы разрабатываются как набор вычислительных процессов, взаимодействующих друг с другом и работающих одновременно, то есть параллельно.

Параллельные программы могут физически исполняться либо последовательно на единственном [процессоре](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) — перемежая по очереди шаги выполнения каждого вычислительного процесса, либо параллельно — выделяя каждому вычислительному процессу один или несколько процессоров (находящихся [рядом](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%BD%D0%BE%D1%81%D1%82%D1%8C) или [распределённых](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) в компьютерную [сеть)](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C).

Главная сложность при планировании архитектуры параллельных вычислений – обеспечение правильной и корректной последовательности взаимодействий между различными параллельными вычислительными процессами, а также распределение ресурсов между процессами и их обмен.

Программа, выполняемая параллельно, примерно выглядит следующим образом:

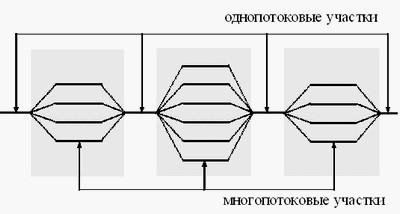


Рисунок 2 – структура параллельной программы.

Как видно на рисунке – определенные области программы выполняются в несколько потоков. При завершении этапа вычислений, нити потоков соединяются, происходит синхронизация, далее вычисления разделяются снова.

При написании параллельных программ требуется особое внимание работе с данными. При неправильной проектировании программы возникают следующие ошибки:

* Гонка данных:

Эта ошибка возникает в случае одновременной попытки изменения разыми процессами некоторой общей области данных. Конечное значение будет неправильным и будет зависеть от того, какой процесс взаимодействовал с ней последним. В случае, когда несколько задач попытаются обновить один и тот же ресурс данных, такое состояние «гонок» называют «гонкой»данных (data race). Для решения этой проблемы следует ввести «критические секции», то есть области, в которых в момент времени должен работать только один процесс, а остальные должны находиться в ожидании в этот момент времени, пока взаимодействующий с областью данных процесс не закончит свою работу. После чего начать выполнять критическую секцию сможет другой поток. Такая схема работы называется "блокировкой" потоков.

* Дедлок:

Клинч, дедлок (deadlock) - разные названия одной из самых серьезных проблем, возникающих при параллельном программировании. Такая ошибка может возникнуть в том случае, когда несколько потоков пытаются получить общие ресурсы в один момент времени. При клинче каждый из потоков успевает захватить один из общих ресурсов. Для окончания работы каждому потоку необходимы другие ресурсы, захваченные другими потоками. В результате, никто из потоков не может завершить свою работу, все стоят в очередях, которые не двигаются, - работа замирает – приложение"зависает". Это худшее, что может случиться с приложением.

Способы синхронизации параллельного взаимодействия:

* Взаимодействие через разделяемую память: на каждом процессоре мультипроцессорной системы запускается поток исполнения, который принадлежит одному процессу. Потоки обмениваются данными через общий для данного процесса участок памяти. Количество потоков соответствует количеству процессоров.

* Взаимодействие c помощью передачи сообщений: на каждом процессоре многопроцессорной системы запускается однопоточный процесс, который обменивается данными с другими процессами, работающими на других процессорах, с помощью сообщений. Процессы создаются явно, путем вызова соответствующей функции операционной системы, а обмен сообщениями — с помощью библиотеки (например, реализация протокола [MPI)](https://ru.wikipedia.org/wiki/Message_Passing_Interface), или с помощью средств языка.

* Гибридный способ: на многопроцессорных системах с распределённой памятью ([DM-MIMD)](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%BA%D1%81%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D1%8F_%D0%A4%D0%BB%D0%B8%D0%BD%D0%BD%D0%B0#DM-MIMD_.28distributed_memory_MIMD.29), где каждый узел системы представляет собой [мультипроцессор](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) с общей памятью ([SM-MIMD)](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%BA%D1%81%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D1%8F_%D0%A4%D0%BB%D0%B8%D0%BD%D0%BD%D0%B0#SM-MIMD_.28shared_memory_MIMD.29), можно

использовать гибридный метод программирования[[4].](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F#cite_note-RedBook.E2.80.941999.E2.80.94.E2.80.945-4) На каждом узле системы запускается многопоточный процесс, который распределяет [потоки](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F) между процессорами данного узла. Обмен данными между потоками на узле осуществляется через общую память, а обмен данными между узлами — через передачу сообщений. В этом случае количество процессов определяется количеством узлов, а количество потоков — количеством процессоров на каждом узле. Гибридный способ программирования более сложен (требуется особым образом переписывать параллельную программу), но наиболее эффективен в использовании аппаратных ресурсов каждого узла многопроцессорной системы.

Разумеется, в такой системе можно также использовать и исключительно метод передачи сообщений, то есть запустить на каждом процессоре каждого узла отдельный процесс. В этом случае количество процессов (и потоков) будет равно количеству процессоров на всех узлах. Этот способ проще (в параллельной программе надо только увеличить количество процессов), но является менее эффективным, так как процессоры одного и того же узла будут обмениваться друг с другом сообщениями, словно они находятся на разных машинах.

2.2 Краткий обзор используемой параллельной технологии

OpenMP (Open Multi-Processing) — открытый стандарт

для [распараллеливания программ](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC) на языках [Си,](https://ru.wikipedia.org/wiki/%D0%A1_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) [Си++.](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%2B%2B) Это совокупность директивдиректив компилятор[а,](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%80%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) библиотечных [процедур](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и [переменных окружения](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5_%D1%81%D1%80%D0%B5%D0%B4%D1%8B) ,для программирования [многопоточных приложений](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C) на [многопроцессорных системах](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D0%BC%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%BD%D0%BE%D1%81%D1%82%D1%8C) с [общей памятью.](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D1%8F%D0%B5%D0%BC%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C) OpenMP работает максимально абстрактно и не вовлекает программиста в сложные задачи, вроде разделения ресурсов и взаимодействия между процессами. Позволяет без особых усилий и затрат времени распараллелить уже существующую программу, путем добавления OMP-директив компилятора. Значительная часть функциональности OpenMP реализуется при помощи директив. Они должны быть явно вставлены программистом, что позволит выполнять программу в параллельном режиме.

Объектом действия большинства директив является один оператор или блок, перед которым расположена директива в исходном тексте программы.

3. Архитектура клиент-сервер

Клиент-сервер – сетевая архитектура, в которой сетевое взаимодействие разделено между двумя сторонами – клиентом и сервером. Клинт выступает в роли заказчика, запрашивая данные у сервера, а сервер, в свою очередь, в роли поставщика, поставляю данные клиентам. Клиент и сервер –это компьютерные программы. Они взаимодействуют между собой через компьютерную сеть посредством сетевых протоколов и находятся на разных вычислительных устройствах, но могут также выполняться и на одном.

Сервер – программа, принимающая запросы от клиентов и предоставляющая

им данные в зависимости от полученного запроса.

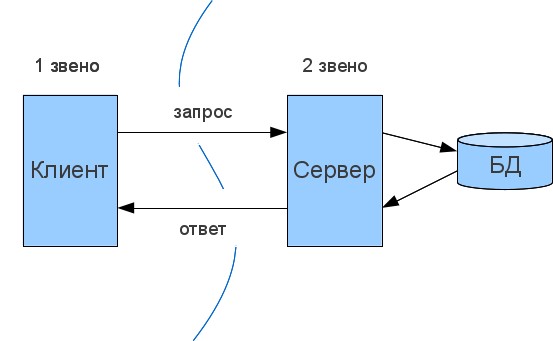
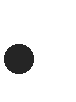


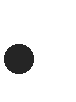
Рисунок 3 – схема архитектуры клиент-сервер

Преимущества:

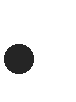
Нет необходимости дублировать код в клиентских приложениях.



Требования к клиентским устройствам снижаются, так как все вычисления производятся на сервере.

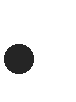


Данные клиентов хранятся на сервере, который защищен лучше, чем большинство клиентов. Зачастую, на сервере организована система управления полномочиями, распределяющая уровни доступа к данным между клиентами с различными правами.



Недостатки:

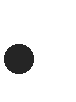
Отказ в работе сервера может сделать вывести из строя всю сеть. Неработоспособным сервером следует считать сервер, производительности которого не хватает на обслуживание всех клиентов, либо сервер, который выключен (вышел из строя).



Поддержка работы такой система требует специалиста – системного администратора.



Стоимость оборудования выше.



Существует несколько видов архитектуры клиент-сервер, это такие виды как двухзвенная или трехзвенная. Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. Т.е. сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса.

Архитектура дипломной работы является двухзвенной.

Зачастую, выполнение запроса занимает времени больше, чем время, затрачиваемое на пересылку запроса и результата. Поэтому в серверах применяются системы распределения транзакций между доменами сервера. Транзакция - это набор команд, которые либо выполняются все или не выполняется ни одна. Домен - это элемент сервера. Нередко сервер на физическом уровне представляет собой не один, а группу компьютеров, объединенных в одну систему для обеспечения каких-либо операций.

Сервер способен принимать запросы от нескольких клиентов сразу и работать с ними. Это называется многопользовательским режимом, но на самом деле, сервер обрабатывает запросы последовательно. Если приходит несколько запросов в один момент, то они становятся в очередь. Иногда запросы могут иметь различный приоритет. Запросы с более высоким приоритетом должны выполняться раньше чем запросы с более низким приоритетом.

Операционная система Windows является многозадачной и

многопоточной. Это означает, что в системе может одновременно быть запущено на исполнение несколько задач. Многопоточность позволяет запускать несколько задач, каждая из которых будет обрабатывать отдельный запрос, тем самым повышая производительность сервера.

Существует концепции построения системы клиент-сервер:

1. Слабый клиент - мощный сервер. В такой концепции всеми вычислениями оперирует только сервер. После окончания вычислений он посылает готовый результат, не требующий дополнительной обработки. Клиент только ведет диалог с пользователем: составляет запрос, отсылает запрос, принимает запрос и выводит информацию на экран (на принтер, в файл).
2. Сильный клиент - часть обработки информации перепоручается клиенту, например предварительная обработка данных.

При частичной обработке данных на клиенте "время ожидания" меньше. Меньше оно за счет упрощения запроса и времени его выполнения.

Отсюда меньше ожидание в очереди для исполнения запроса. Безусловно, что время конечной обработки на клиенте может быть немного выше за счет разности в производительности сервера.

Отсюда может несколько возрастать время ожидания. В конечном счете, это все равно выгоднее, т.к. время ожидания в очереди запросов на сервере меньше. Многие серверы не выдерживают нагрузки ("напора" запросов) и просто "подвисают". В таком случае есть два альтернативных способа: увеличение производительности и перенос части операций над данными на клиента. Как правило, увеличение производительности значительно более дорогостоящая операция, и тоже, в своем смысле, конечная. Остается только "разгрузка" сервера и перенос части обработки данных на клиента. Нередко в ряде организаций в качестве сервера сознательно используют устаревшие компьютеры. На них хорошо размещать файловые хранилища, принт-серверы (к нему подключается офисный принтер), WEB-серверы (Интернет-серверы), небольшие базы данных (серверную часть). Это оправдано с экономической точки зрения.

Применение самых мощных компьютеров в качестве серверов целесообразно в банковской сфере, т.к. объем платежей неизменно возрастает. Соответственно, возрастает необходимый объем вычислительных ресурсов. Здесь оправдана "борьба" за единицы процентов увеличения производительности.

1. Программирование для мобильных платформ

4.1 Общие сведения о мобильной разработке

Разработка приложений для мобильных платформ является одним из ведущих направлений в мире IT в настоящее время. Android – мобильная операционная система, основана на [ядре Linux](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux) и собственной реализации виртуальной машины [Java](https://ru.wikipedia.org/wiki/Java_Virtual_Machine) от [Google.](https://ru.wikipedia.org/wiki/Google_(%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F)) Но несмотря на то, что платформа основана на ядре Linux, она имеет много уязвимостей. Платформа развивается со стремительной скоростью и по возможностям не уступает даже ПК, В настоящее время разработка под эту мобильную ОС является одной из основных тенденций в мире разработки вообще. ОС предоставляет пользователям возможность работы с сетью интернет, службами геолокации, Wi-Fi, Bluetooth и протоколы передачи данных через сотовую сеть (GPRS, EDGE, 3G и др.).

Android содержит встраиваемый браузер на базе WebKit - того же механизма с открытым исходным кодом, который лежит в основе браузера Safari для iOS.

Приложения, написанные для Android, состоят из следующих элементов:

* + Активити(Activity) – это экраны, которые видит пользователь, при работе с приложениями.
  + Фрагменты (Fragment) - это модули, содержащие графический интерфейс, которые встраиваются в активити. За счет использования фрагментов, приложения проще разрабатывать, т.к фрагмент имеет собственную логику и может быть переиспользован. Зачастую, активити может содержать несколько фрагментов.
  + Сервисы (Services) - сервисы отвечают за работу функций приложения в фоне. Они могут продолжать работать, даже когда пользователь не взаимодействует с приложением.
  + Сервисы применяются при длительных операциях взаимодействия с сетью, мониторинга ресурсов или для фоновой проверки обновлений.
  + Источники данных (Content providers) - источник данных можно представить себе как сервер баз данных. Его задача - управление доступом к хранящимся данным, таким как база данных SQLite. Если приложение совсем простое, источник данных создавать не обязательно. Если вы пишете более сложное приложение или приложение, в котором к данным обращается несколько действий или приложений, источник данных служит средством организации доступа к вашей информации.
  + Приемники (Broadcast receivers) - Android-приложение может запускаться для обработки элемента данных или реагирования на события, например, на получение текстового сообщения.

Приложение для Android обязательно должно иметь файл AndroidManifest.xml, в котором содержится основная информация о приложении и его конфигурации для правильной установки приложения на устройстве пользователя. В манифесте описывается такая информация, как: необходимые имена классов и типы событий, которые может обрабатывать приложение, разрешения, требуемые для его работы. Так, если приложению нужен доступ к сети – например, чтобы загрузить файл, - соответствующее разрешение должно быть явно указано в файле манифеста. Такая защита путем декларирования помогает уменьшить вероятность повреждения устройства по вине некорректно написанного приложения. Стоит отметить, что система разрешений изменилась с выходом Android 6.0 и разрешения,

запрашиваемые приложением не отображаются при установке, а запрашиваются в режиме реального времени. Т.е. приложение должно иметь минимум разрешений, необходимых для работы, т.к. пользователь вправе отказать приложению на запросы некоторых разрешений, например если он не хочет давать доступ камере к своему местоположению.

Для разработки под Android необходима IDE и Android SDK, знание языка программирования Java и языка XML для создания макетов интерфейсов. Неотъемлемой частью разработки является запуск приложений, поэтому также необходимо реальное устройство, либо эмулятор. Особо популярны два эмулятора: Genymotion, Android Studio Emulator 2.0.

* + Genymotion – это быстрый эмулятор Android, который включает в себя настроенные образы Android (x86-с аппаратным ускорением OpenGL), идеальный для тестирования приложения. К минусам этого эмулятора можно отнести нестабильность: эмулятор запускается далеко не с первого раза, а в некоторых случаях количество неудавшихся попыток запуска достигает 10!

* + Android Studio Emulator 2.0 – это относительно новый сверхбыстрый эмулятор, выпущенный компанией Google и поставляемый вместе со средой программирования Android Studio 2.0.

Разработчики уверяют, что по сравнению со старым эмулятором, новый работает до 10 раз быстрее и также приложения устанавливаются на него до 3 раз быстрее.

К минусам этого эмулятора можно отнести требовательность к ресурсам: для комфортной разработки в Android Studio и отладки на этом эмуляторе нужно не менее 8гб оперативной памяти и процессора не менее чем с 4 ядрами.

При работе над дипломным проектом используются все три варианта тестирования: два эмулятора и реально устройство (Google Nexus 5).

Разработчик приложений должен хорошо знать особенности ОС Android и помнить о сегментации устройств и различных версий ОС. Существуют некоторые трудности, которые нужно учитывать при разработке:

1. Приложение требует для установки в два раза (или даже в четыре) больше места чем оригинальный размер приложения.
2. Скорость работы с файлами на встроенной флэшке падает в десятки раз при уменьшении свободного места.
3. Каждый процесс может использовать до 16 Мб (иногда 24 Мб) оперативной памяти.
4. Зачастую один и тот же код или элемент интерфейса ведет себя поразному на разных устройствах и версиях Android. Несмотря на разработку библиотеки поддержки Support Library, этот пробел не удается устранить полностью.
5. Всегда найдется специфическое устройство, на котором приложение работает совершенно не так, как задумано. Из этого вытекает пункт 6.
6. Сложность отладки. Как правило, средний разработчик не может протестировать приложение на всех доступных устройствах по причине их многообразия и собственно, отсутствия в наличии их у разработчика.

Существует два способа разработки для мобильных устройств:

написание нативных приложений и использование кроссплатформенного движка.

Нативным способ – написание приложений на родной языке программирования для платформы. Для Android это Java и Android SDK.

Такой способ наиболее гибкий и удобный, и производительный. Но недостаток его в том, что целевая платформа только одна. Нативный способ разработки стоит выбирать в том случае, если перенос приложения на другие платформы не планируется.

Кроссплатформенный способ – при выборе этого способа, приложения разрабатываются сразу для нескольких платформ (Android, iOS, WP). Такие приложения имеют набор общей логики, общий интерфейс, но мелкие детали приходится дорабатывать в зависимости от платформы. Кроссплатформенная разработка более подвержена ошибкам, чем нативный способ. Также, количество документации к кроссплатформенным средствам существенно ниже, чем к нативным. К крассплатформенным средствам разработки можно отнести такие как Xamarin от Microsoft, Apache Cordova, Phonegap, также

Microsoft сделала недавно возможной кроссплатформенную разработку в IDE Visual Studio.

4.2 Архитектурный паттерн MVP

Архитектурные шаблоны являются важнейшей частью ПО. Они помогают сохранить код в чистоте, сделать его расширяемым и тестируемым. Шаблон MVP позволяет отделить уровень представления от уровня логики, для того что бы поведение приложения не зависело от его конкретного внешнего вида. Android не определяет зоны ответственности между компонентами приложения, поэтому вся логика работы с UI и данными приложения описана внутри одной Activity, что не позволяет сделать приложение расширяемым и легко тестируемым. Использование MVP позволяет решить эту проблему. Есть много разных подходов для реализации MVP, но не зависимо от выбранного решения должны сохранятся три компонента:

1. Presenter выступает в качестве посредника между View и Model. Он извлекает данные из модели и передает их во View. Но в отличие от типичного MVC, он также решает, что нужно делать, когда вы взаимодействуете с View.

Презентер отвечает за:

* + Загрузку моделей;
  + Сохранение ссылки на модель и состояния представления;
  + Форматирование того, что должно быть отображено на экране, и указание представлению отобразить это;
  + Определение необходимых действий, когда получены входные события от представления.

1. View, как правило, реализуется в Activity и Fragment, которые содержат ссылку на презентер. Единственное, что делает View, это вызывает методы презентера при каком-либо действии пользователя.

Представление отвечает за:

* + Создание экземпляра презентера и механизм его присоединения/отсоединения;
  + Оповещение презентера о важных для него событиях жизненного цикла;
  + Сообщение презентеру о входных событиях;
  + Размещение представлений и соединение их с данными;
  + Анимации;
  + Отслеживание событий;
  + Переход на другие экраны

1. Model рассматривается в качестве поставщика данных, которые будут отображаться во View. Модель — это набор бизнес-логики.

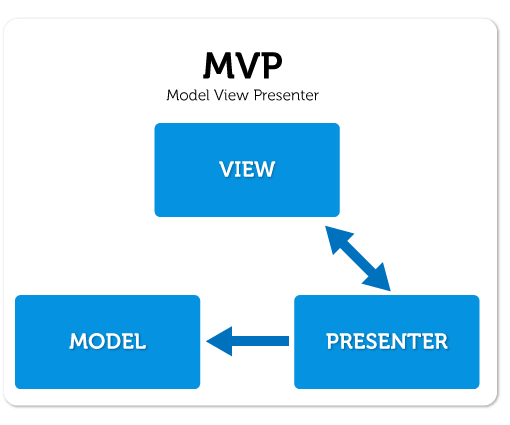


Рисунок 4 – взаимодействие компонентов приложения в MVP

Также использование паттерна MVP почти полностью решает вопрос с сохранение состояния активити при ее пересоздании. В традиционном подходе все данные хранятся в активити, а при пересоздании активити она разрушается и данные теряются. В паттерне MVP активити не хранит никаких данных. При создании активити в первый раз создается презентер для нее. Когда представление доступно для взаимодействие, презентер прикрепляется к активити, когда недоступно – открепляется. Тем самым избавляя приложения от главных проблем – утечек памяти, сохранения данных и привязки асинхронных операций к активити. Как известно, в ОС Android асинхронная задача AsyncTask имеет неявную ссылку на активити, ее запустившую и при после пересоздания активити AsyncTask хранит ссылку на несуществующую активити и в итоге результат асинхронной операции придет в никуда и случится утечка памяти. С приходом MVP эта проблема также решена. Можно запускать асинхронные операции в модели.

1. Состав программ и основные алгоритмы

Для реализация программы, выполняемой на стороне сервера будет использоваться язык C++ и среда разработки Microsoft Visual Studio 2013, параллельная технология OpenMP. Для построения мобильного клиентского приложения на Android используется Android SDK, среда разработки Android Studio 2.1, библиотека Okio OkHttp для выполнения http запросов, библиотека GraphView для отрисовки графиков. Для реализации сервера был выбрал скриптовый язык PHP 5.5, база данных MySQL.

* 1. Программа для моделирования ad-hoc сетей состоит из 4-х классов:
     + 1. World – класс общей сферы, содержит в себе список узлов, препятствие, операции для работы с сетью (графом). Операция для генерации узлов, генерация препятствия, некоторые геометрические операции, такие как разность векторов, умножение векторов, модуль вектора,
       2. Нахождение расстояние между двумя точками, расстояние от точки до прямой в пространстве, функция для проверки возможности установления соединения между двумя узлами. Также есть функции для подсчета компонент связности сгенерированной сети.
       3. Elem – базовый класс узла в общей сфере. Содержит декартовы координаты, радиус узла. При создании объекта класса *Elem*, в конструктор передаются сгенерированные случайным образом зенитный и азимутный углы, кратчайшее расстояние от начала координат то узла. Далее, на основе этих величин осуществляется преобразование в декартовы координаты. Сферические координаты в классе не хранятся, только декартовы.
       4. Node, Obstacle – классы, производные от *Elem*. Класс *Node* содержит список ребер и операцию для проверки наличия узла в списке ребер.

* + 1. Генерация узлов

Класс World содержит операции для генерация узлов.

Рассмотрим два случая:

1. Общая сфера не содержит препятствий. Тогда алгоритм генерации узлов будет следующий:

* 1. Генерируем три случайные сферические координаты.
  2. Создаем новый узел и добавляем его в список узлов графа.

2. Общая сфера содержит препятствие.

Алгоритм:

* 1. Генерируем три сферические координаты.
  2. Создаем временный узел и задаем ему эти координаты.
  3. Если препятствие и узел не пересекаются, добавляем узел в граф.

Генерация происходит до тех пор, пока количество сгенерированных узлов меньше заданного кол-ва элементов для генерации.

Два элемента пересекаются, если расстояние между ними меньше радиуса самого элемента. Последовательность шагов:

1. Вычисляется расстояние между двумя точками по форму

*(3)*

Если вычисленное расстояние меньше радиуса элемента, то возвращается значение true. Это означает что элементы пересекаются, иначе false.

Алгоритм генерации препятствия следующий:

* 1. Генерируем три случайные сферические координаты.
  2. Создаем новое препятствие и добавляем его в список препятствий графа.

* + 1. Установка соединения между узлами сети

Одной из главных целей работы является поиск и установка соединения между узлами. Соединение устанавливается, если расстояние между двумя узлами меньше радиуса узла *r***.** Наличие соединения означает возможность обмена информацией между этими узлами. Все множество соединенных узлов будет образовывать граф.

Таким образом, необходимо проверить все пары вершин графа на возможное наличие связи между ними.

Задача поиска разделяется на 2 случая:

* + - 1. Граф имеет препятствие в виде сферы.
      2. Препятствие отсутствует.

В первом случае узлы графа уже сгенерированы так, чтобы их координаты лежали за пределами радиуса препятствия. Препятствие может занимать 0%, 10%, 20% и 50% это объема общей сферы.

Также для первого случая необходимо иметь процедуру для вычисления расстояние от точки (центра препятствия) до прямой (вектора между двумя узлами).

Величина векторного произведения двух 3-мерных векторов равна площади параллелограмма, построенного на них. Однако эта площадь также равна произведению основания на высоту параллелограмма, а длина высоты - искомая дистанция d (P, L). Пусть и .

Представим, что точка и точка – центры узлов. Точка P – центр препятствия. Нахождение расстояние между точкой и прямой по формуле

(4):

*(4)*

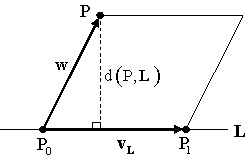


Рисунок 5 - Параллелограмм

Сам алгоритм установления соединений между узлами сети заключается в следующем:

1. Перебираются все пары вершин графа.
2. Если расстояние между двумя узлами (*node1* и *node2*), рассматриваемыми на текущей итерации, меньше r, то выполняется переход к следующему шагу алгоритма.
3. Если между узлами *node1* и *node2* еще нет соединения, то есть в списке ребер *node1* нет пути к узлу *node2*, то переходим к следующему шагу.

Дальнейшие действия алгоритма зависят от того, есть ли в графе препятствие.

1. Если препятствие есть, то необходимо проверить, мешает ли оно установлению соединения между узлами. Если да, выполнение алгоритма продолжается с другой парой узлов. Если нет, то устанавливается соединение между узлами, рассматриваемыми в данный момент.

Если же препятствия нет и предыдущие пункты алгоритма истинны, то соединение устанавливается.

В список ребер *node1* добавляется узел *node2* и, соответственно, в список *node2* добавляется узел *node1*.

5.2 Распараллеливание вычислений

Основные вычисления в программе – генерация новых графов и обработка их статистических данных. Эти вычисления будем распараллеливать с помощью технологии OpenMp.Целью использования параллельных технологий является уменьшение времени, затрачиваемого на работу программы. *const int size = 100; const float node\_radius = 1; const float obst\_radius = 5; const float world\_radius = 50; const int nodes\_count = 200;*

*int**main****()***

***{****#pragma omp parallel for reduction (+:comps) reduction(+:nodes) firstprivate(world)*  *for (int iters = 0; iters < size; ++iters)*

*{*

*world.generateObstacle(); world.generateNodes(nodes\_count); world.checkEdges();*

*disper[iters] = world.intersectionsCount(); nodes += world.intersectionsCount(); comps += world.componentsCount(); world.clearWorld();*

*}*

Здесь *#pragma omp parallel for* – директива компилятору, обозначающая начало параллельной секции для цикла *for*.

reduction (+:comps) reduction(+:nodes) - обозначают переменные, с которыми в цикле производятся reduction-операции (суммирование). При выходе из цикла, данные операции производится над копиями переменных во всех нитях, и результат присваивается оригинальным переменным.

firstprivate(world) - приватные копии переменной при входе в параллельную область инициализируются значением оригинальной переменной.

5.3 Организация сервера

В дипломной работе сервер, по сути, является посредником между вычислительной программой и клиентом, а также оберткой над базой данных.

Серверный скрипт состоит из 4 классов:

1. Request – класс, описывающий набор параметров входящего клиентского запроса. Список параметров:

$iterationsCount – количество итерация тестирования

$nodeRadius – радиус узла Ad-hoc сети

$obstacleRadius – радиус препятствия в Ad-hoc сети

$worldRadius – радиус сферы, в которой проводится моделирование

$nodesCount – количество узлов сети

$hasObstacle – флаг, показывающий, есть ли препятствие в сфере

Также класс содержит некоторые вспомогательные методы, такие как геттеры, сеттеры, конструктор и метод, преобразующий набор параметров в аргументы командной строки.

1. Response – класс, описывающий набор параметров результата

$nodesAvg – математическое ожидание количества ребер в сети

$dispersion – дисперсия математического ожидания количества ребер в сети

$compsAvg – математическое ожидание количества компонент связности в сгенерированной сети.

Также класс содержит некоторые вспомогательные методы, такие как геттеры, сеттеры, конструктор и метод, формирующий Json-объект из набора параметров.

1. DbOperations – класс-обертка над базой данных, содержащий все основные операции.

Методы: findResult($connection, $request) – метод ищет в БД запись, соответствующую параметру $request. $request – перемнная типа Request.

Возвращает Id эксперимента, если такой существует и -1, если нет. insertExperiment($connection, $params, $results) – метод, вставляющий две новые записи в две таблицы базы данных – параметры эскперимента и его результат.

insertResult($connection, $result) – метод добавляет результат эксперимента в таблицу БД и возвращает ID вставленной записи.

1. Server – php файл, реализующий обработку входящих запросов и выдачу результатов.

5.3.1 Алгоритм обработки запросов

Сервер принимает GET запросы с параметрами запроса, в котором содержится набор параметров. Сервер проверяет, является ли запрос корректным, т.е все ли требуемые для проведения эксперимента параметры содержатся в параметрах запроса. Далее сервер проверяет, существует ли кэшированный эксперимент в базе данных: если да, то формируется JSONтело ответа и отправляется обратно клиенту, если нет, запускается исполняемый файл с аргументами командной строки, состоящими из параметров, полученных вместе с запросом. После завершения работы исполняемого файла, результат кэшируется в базу данных, а результат эксперимента отправляется клиенту.

Формат Json-ответа:

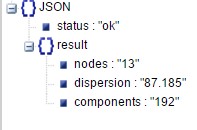


Рисунок 6 – формат Json ответа сервера

Схема базы данных:

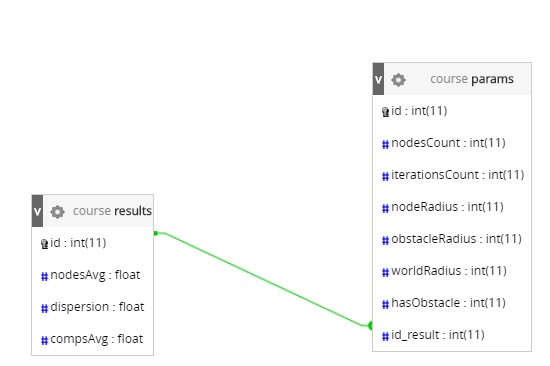


Рисунок 7 – схема базы данных

На сервере и клиенте всего по одной таблице. Таблица Params и таблица

Result.

5.4 Структура проекта клиентского Android-приложения

В качестве архитектурного паттерна клиентского приложения используется паттерн MVP (model, view, presenter).

Структура проекта в Android Studio:

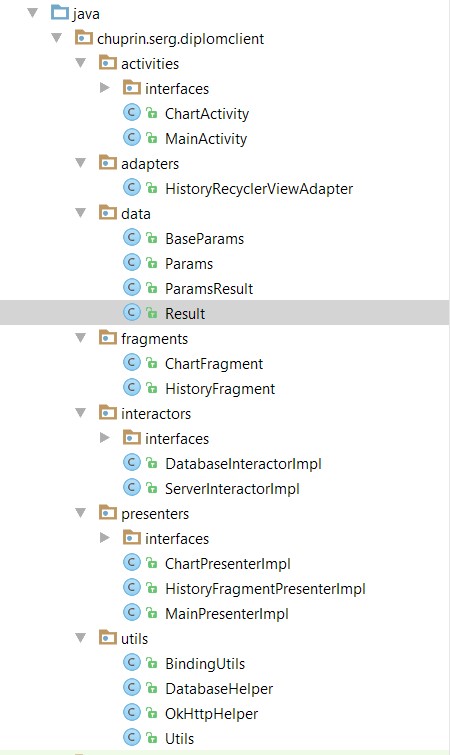


Рисунок 8 – структура проекта в Android Studio

Основой бизнес-логики приложения являются классы работы с базой данных(DatabaseInteractorImpl), общения с сервером(ServerInteractorImpl).

Пакет Activities:

ChartActivity – активити(представление), в которой отображаются графики. MainActivity – стартовый экран приложения(представление), в котором отображен список параметров для проведения экспериментов.

Пакет Data:

Params – класс-обертка над параметрами эксперимента.

Result – класс-обертка над результатом эксперимента.

Пакет Presenters:

ChartPresenterImpl – презентер для ChartActivity, отвечающий за преобразование данных для отображения в представлении, получаемых из базы данных.

MainPresenterImpl – презентер для MainAcitivty.

Пакет Utils:

Содержит вспомогательные классы с некоторым набором полезных функций.

OkHttpHelper – вспомогательный класс, содержит HTTP клиент для выполнения сетевых операций. Класс отвечает за формирование параметров запросов.

DatabaseHelper – класс, отвечающий за создание базы данных для кэширования экспериментов. Содержит поля таблиц базы данных.

5.4.1 Логика работы клиентского приложения

На главном экране есть список параметров для проведения эксперимента. После того как пользователь вводит параметры и нажимает кнопку отправки, клиент проверяет, существует ли в локальном кэше результат эксперимента для введенных параметров эксперимента, если да, то пользователю сразу же выводится сообщение с результатами эксперимента. Если же нет, то устанавливает соединение с сервером и отправляется HTTP POST запрос на сервер со списком параметров. При получении ответа от сервера, клиент кэширует в локальную базу данных результат эксперимента, с соответствующим ему списком параметров, и так же выводит сообщение пользователю с результатом эксперимента.

При наличии достаточного количества проведенных экспериментов, пользователь может нарисовать графики с зависимостями результатов эксперимента от различных входных параметров (количество узлов, радиус, радиус сферы и т.д). Всего существует три графика:

График математического ожидания количества соединений в сгенерированной сети, график дисперсии математического ожидания в сгенерированной сети и график математического ожидания количества компонент связности в сгенерированной сети.

6 1 Демонстрация работы мобильного клиента

После запуска клиент встречает пользователя стартовым экраном со списком всех доступных параметров для моделирования Ad-hoc сетей, выполненным по гайдлайнам Material Design:

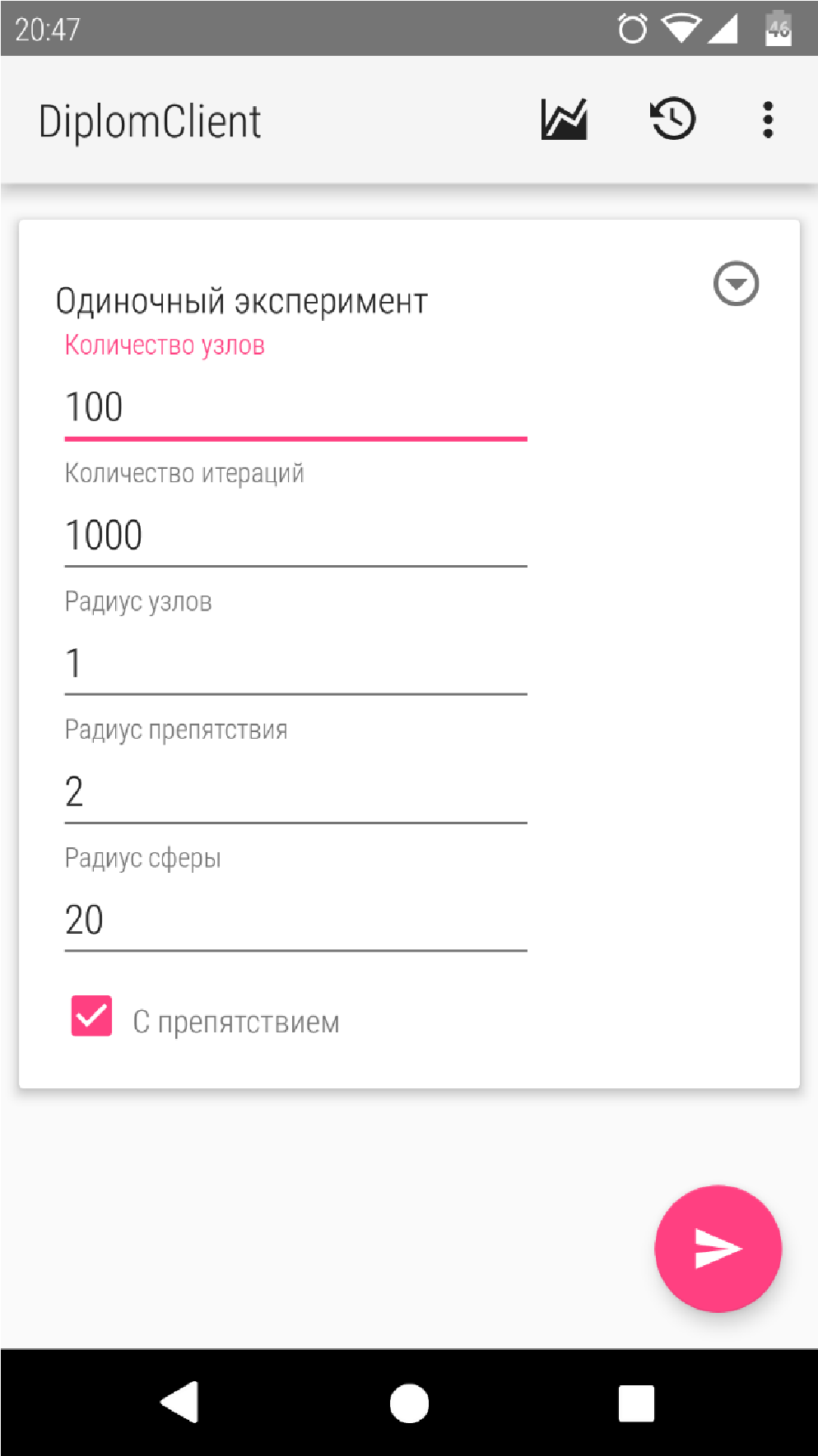


Рисунок 9 – стартовый экран клиентского приложения

Как видно, на экране перечислен список всех доступных параметров Ad-hoc сетей для проведения эксперимента. Внизу кнопка отправки запроса на сервер.

После ответа сервера можно увидеть результаты:

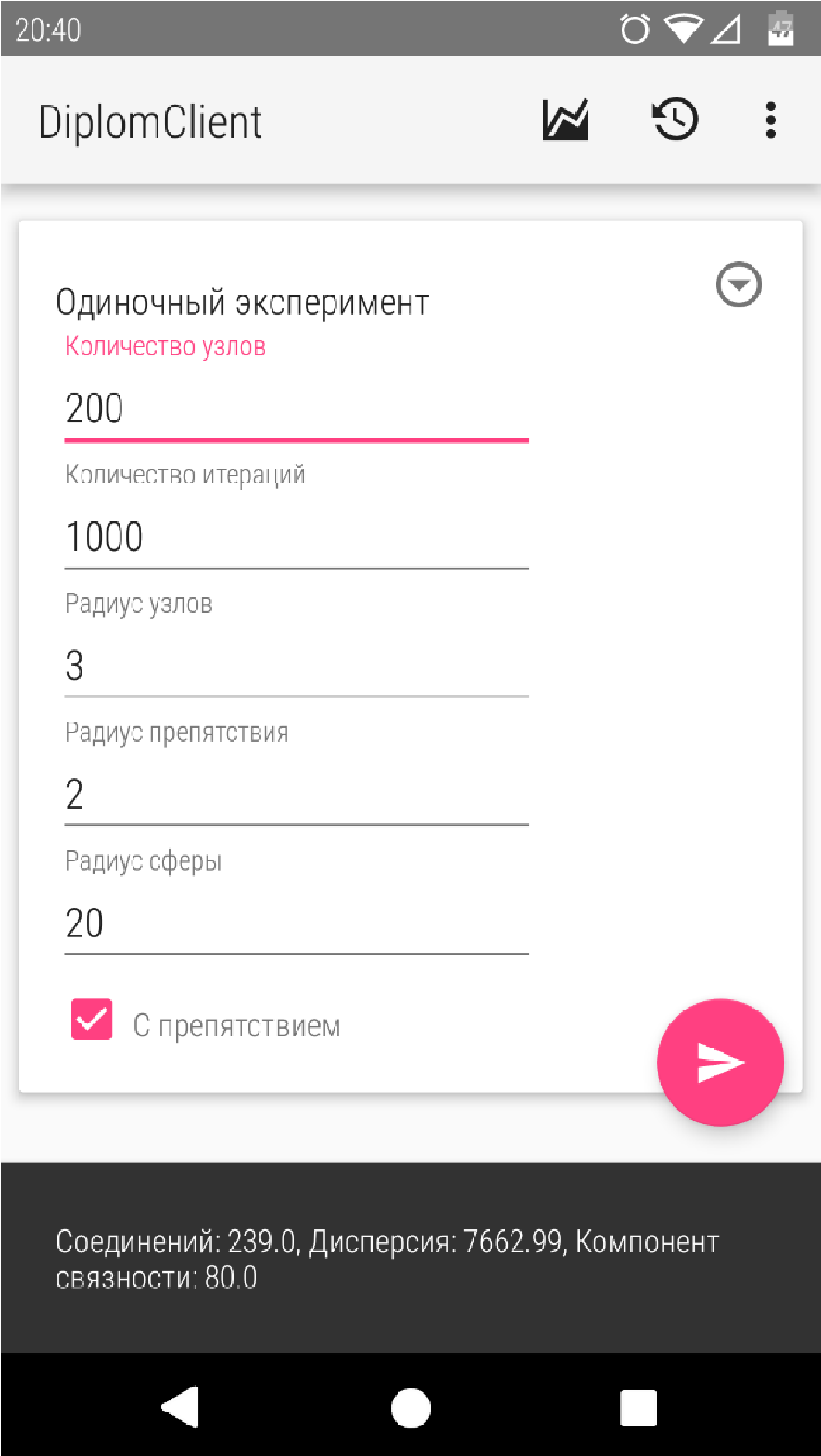


Рисунок 10 – результаты эксперимента в клиентском приложении

Перейдем к экрану с графиками: здесь можно отобразить графики соответствующие набору параметров Ad-hoc сетей:



Рисунок 11 – параметры для рисования графиков на клиентском приложении

Можно запросить графики для каждой характеристики из эксперимента, либо отобразить для какой-либо одной. Также можно добавлять графики к сравнению с уже нарисованными, а можно очистить все и отобразить только один необходимый.

Для удобства просмотра внизу экрана расположены кнопки изменения масштаба. Также, между экранами поддерживается навигация свайпом, либо кликом по табам вверху страницы.

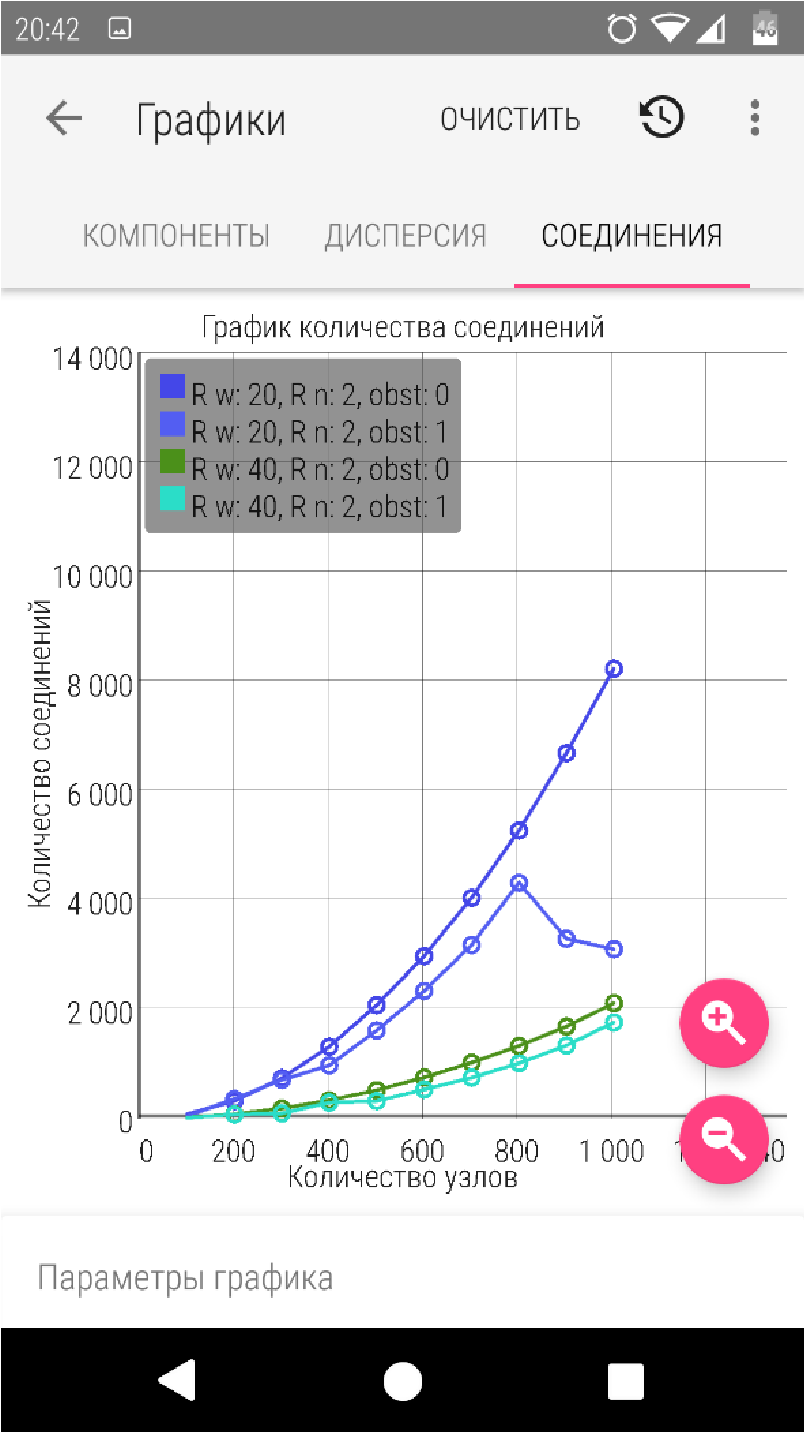


Рисунок 12 – график количества соединений между узлами

Для того, чтобы не запоминать, какие эксперименты пользователь проводил, а какие нет, для удобства использования можно посмотреть историю из списка параметров экспериментов вместе с результатами. Каждый элемент списка может быть развернут по клику:

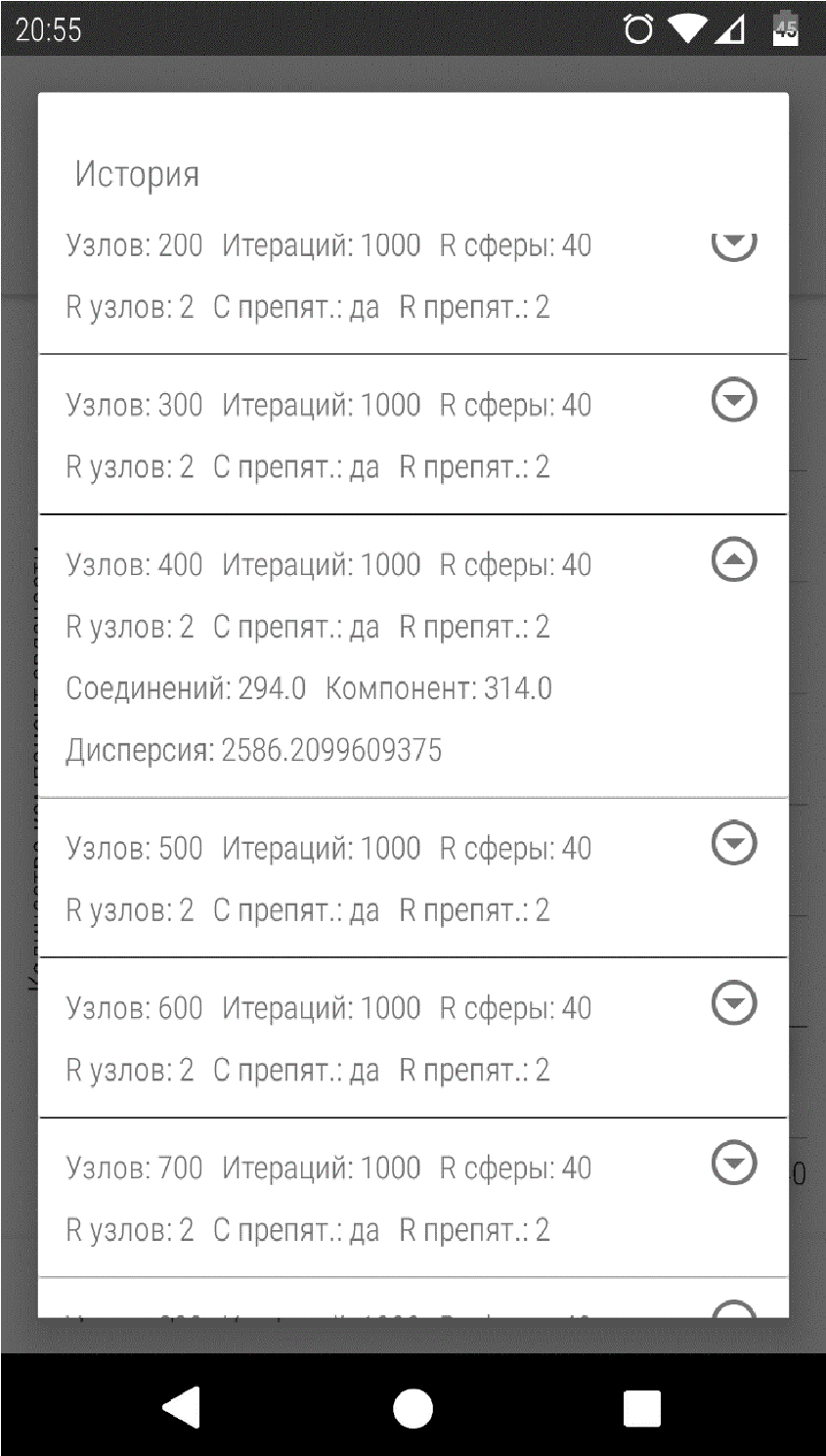


Рисунок 13 – история экспериментов, проведенных на клиенте

Проведем несколько экспериментов: для некоторых параметров получим результаты моделирования сначала без препятствия, затем с препятствием. Интервал количества узлов – от 100 до 1000. Количество итераций тестирования = 1000. Радиус узлов – 2, радиус препятствия – 2.

На первом графике в легенде обозначены две кривые – зеленая (сеть с препятствием) и коричневая (сеть без препятствия). По скриншоту можно судить, что в сети с препятствием соединений несколько меньше.

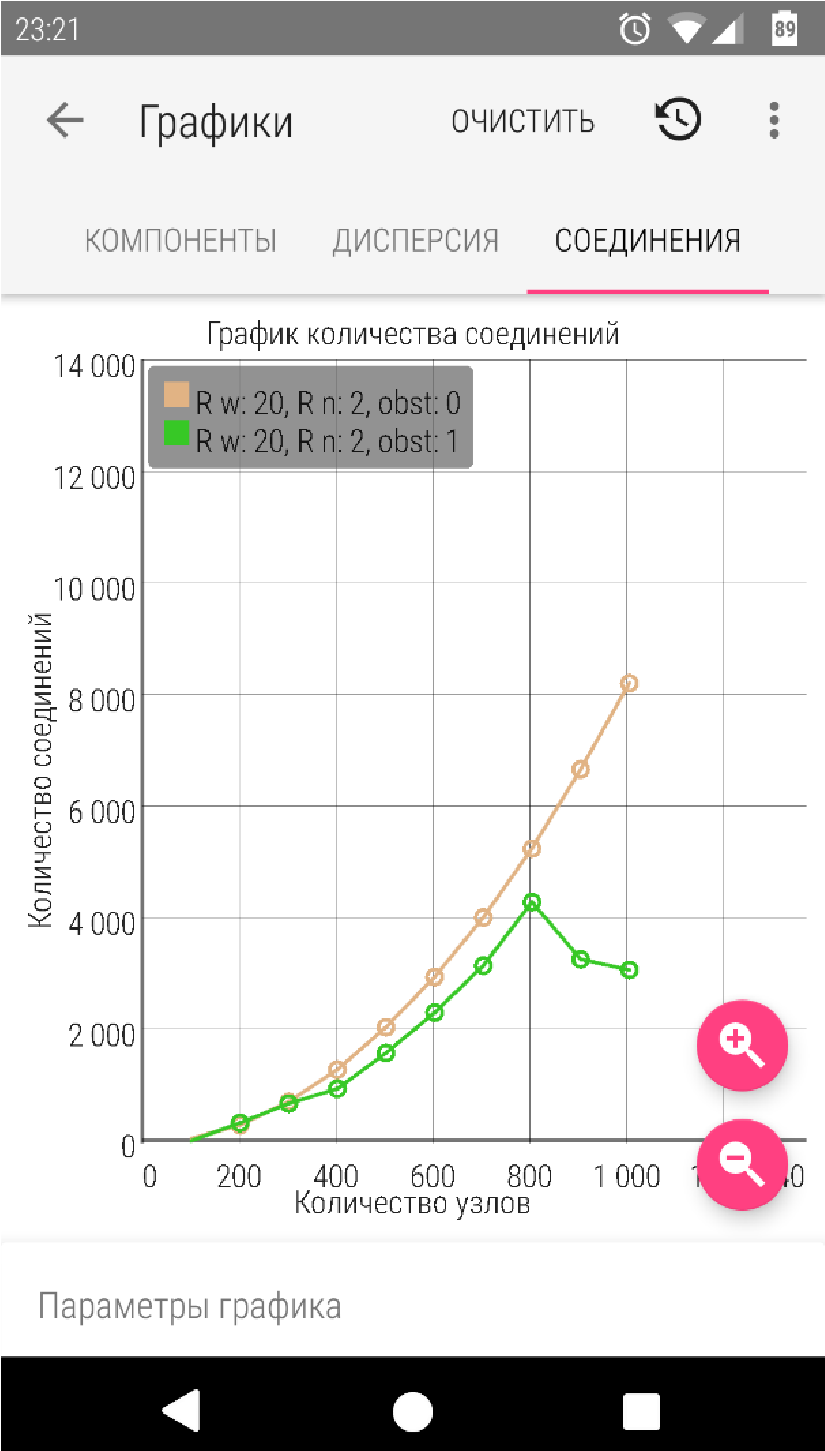


Рисунок 14 – сравнение графиков количества соединений для Ad-hoc сети с препятствием и без

На втором графике на легенде обозначены две кривые – голубая (сеть без препятствия) и коричневая (с препятствием). По графику видно, что в сети с препятствием количество компонент связности немного увеличивается по сравнения с сетью без препятствия, что вполне логично.

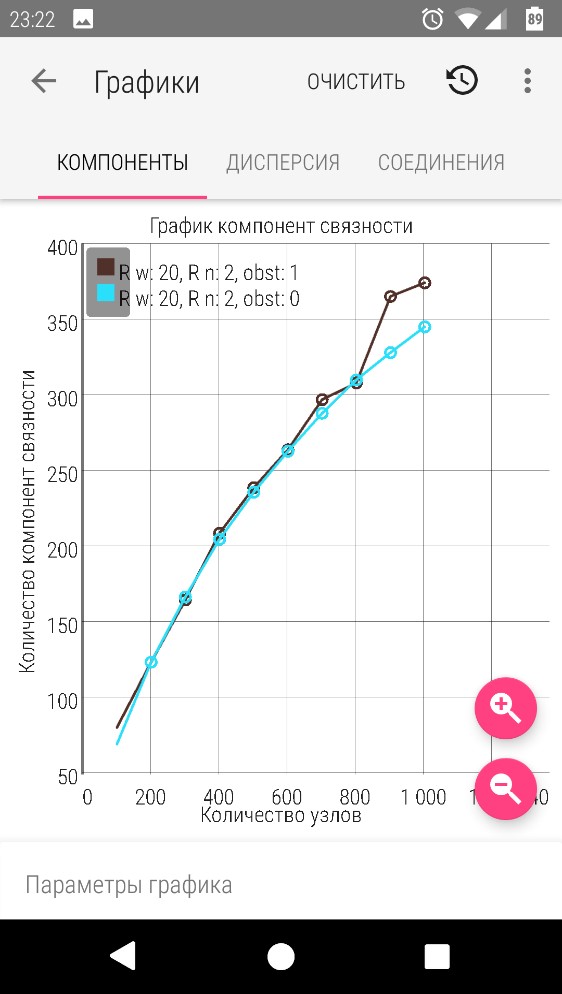


Рисунок 15 – сравнение графиков количества компонент связности для Ad-hoc сети с препятствием и без

ЗАКЛЮЧЕНИЕ

В процессе выполнения дипломной работы было построена полноценная архитектура клиент-сервер, с клиентским приложением для мобильной операционной системы Android. Выбранная архитектура позволяет простым пользователям получить доступ к анализу и моделированию сложных Ad-hoc сетей. В изучены и внедрены современные технологии построения комплексной архитектуры ПО, в частности клиентской и серверной частей. Завершенный дипломный проект является полноценным программным обеспечением, представляющим доступ пользователям к моделированию Adhoc сетей и наглядному исследованию их характеристик. Разработанное мобильное приложение представляет собой быстрый и удобный инструмент для моделирования и анализа беспроводных Ad-hoc сетей в экстренных ситуациях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хагарти Р. Дискретная математика для программистов,// из-во

Техносфера, 2-е изд. 2005г., 400 стр.

1. Таненбаум Э., Уэзеролл Д. Компьютерные сети,// из-во Питер, 5-е изд., 2012г., 960стр.
2. Страуструп Б. Язык программирования C++,// из-во Бином 2-е изд. 2007г.
3. Android API, OpenMp и C++, URL: [https://msdn.microsoft.com/ruru/library/dd335940.aspx](https://msdn.microsoft.com/ru-ru/library/dd335940.aspx) [02.01.2016]
4. PHP документация, URL:<http://php.net/manual/ru/>[4.12.2015]

ПРИЛОЖЕНИЕ А

Исходный код программы для моделирования Ad-hoc сетей: Класс main.cpp:

#include "world.h"

#include <ctime>

#include <omp.h>

const int size = 2000; const float node\_radius = 10; const float obst\_radius = 10; const float world\_radius = 50;

const int nodes\_count = 500;

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Количество итераций = " << size << endl; cout << "Радиус сферы = " << world\_radius << "\nРадиус передатчика = " << node\_radius

<< "\nРадиус препятствия = " << obst\_radius

<< "\nКоличество узлов = " << nodes\_count << endl;

float nodes = 0; float comps = 0;

float disper[size];

float start\_time = clock();

srand(static\_cast<unsigned int>(time(0)));

World world(world\_radius, node\_radius, obst\_radius);

omp\_set\_num\_threads(4);

#pragma omp parallel for reduction(+:comps) reduction(+:nodes) firstprivate(world)//ordered for (int iters = 0; iters < size; ++iters)

{

world.generateNodes(nodes\_count); world.checkEdges();

#pragma omp critical

disper[iters] = world.intersectionsCount();

#pragma omp critical

nodes += world.intersectionsCount();

#pragma omp critical

comps += world.componentsCount();

world.clearWorld();

}

float d = 0; nodes = nodes / size;

for (int i = 0; i < size; ++i)

d += pow(disper[i] - nodes, 2);

cout << "\n\nКоличество соединений: " << nodes << endl;

cout << "Средняя дисперсия: " << d / size << endl;

cout << "Компонент связности: " << comps / size << endl;

cout << "Время работы программы: " << (clock() - start\_time) / 1000 << endl;

system("pause");

return 0;

}

Класс world.cpp:

#include "world.h"

#include <ctime>

#include <cmath>

#include <omp.h>

Elem::Elem(float \_zenit, float \_azimut, float \_r, float \_radius)

{

radius = \_radius;

//transform

x = \_r \* sin(\_zenit) \* cos(\_azimut); y = \_r \* sin(\_zenit) \* sin(\_azimut);

z = \_r \* cos(\_zenit);

}

void Elem::setCords(float \_zenit, float \_azimut, float \_r, float \_radius)

{

radius = \_radius;

x = \_r \* sin(\_zenit) \* cos(\_azimut); y = \_r \* sin(\_zenit) \* sin(\_azimut);

z = \_r \* cos(\_zenit);

}

void Elem::printDec()

{

cout << "x: " << x << " y: " << y << " z: " << z;

}

Node::Node(float \_zenit, float \_azimut, float \_r, float \_radius)

: Elem(\_zenit, \_azimut, \_r, \_radius)

{

visited = false;

}

bool Node::findNode(Node \*node)

{

for (auto it = edges.begin(); it != edges.end(); ++it) if ((\*it) == node)

return true;

return false;

}

Obstacle::Obstacle(float \_zenit, float \_azimut, float \_r, float \_radius)

: Elem(\_zenit, \_azimut, \_r, \_radius)

{}

World::World(float \_world\_radius, float \_node\_radius, float \_obst\_radius)

{

world\_radius = \_world\_radius; node\_radius = \_node\_radius; obst\_radius = \_obst\_radius;

nodes\_count = intersections\_count = 0;

}

World::World()

{

nodes\_count = intersections\_count = 0;

}

World::World(const World& obj)

{

nodes = obj.getNodes(); obstacles = obj.getObstacles();

setIntersectionsCount(obj.getIntersectionsCount()); setNodeRadius(obj.getNodeRadius()); setObstalceRadius(obj.getObstacleRadius());

setWorldRadius(obj.getWorldRadius());

}

World::~World() { nodes.clear();

obstacles.clear();

}

int World::intersectionsCount() const

{

return intersections\_count;

}

void World::clearWorld()

{

nodes.clear(); obstacles.clear();

nodes\_count = intersections\_count = 0;

}

int World::componentsCount()

{

int components = 0;

for (int i = 0; i < nodes.size(); ++i) if (!nodes[i]->visited)

{

DFS(nodes[i]);

++components;

}

return components;

}

void World::DFS(Node \*cur\_node)

{

cur\_node->visited = true; auto it = cur\_node->edges.begin();

while (it != cur\_node->edges.end())

{

if (!(\*it)->visited)

DFS((\*it));

++it;

}

}

void World::getCords(float &zenit, float &azimut, float &r)

{

zenit = rand() % 181; azimut = rand() % 360;

r = rand() % int(world\_radius);

}

float World::vectDist(Elem \*e1, Elem \*e2)

{

return sqrt(pow(e1->x - e2->x, 2) + pow(e1->y - e2->y, 2) + pow(e1->z - e2->z, 2)); }

float World::vectModul(Elem \*e)

{

return sqrt(pow(e->x, 2) + pow(e->y, 2) + pow(e->z, 2)); }

Elem \*World::vectDiff(Elem \*e1, Elem \*e2)

{

Elem \*vect = new Elem; vect->x = e1->x - e2->x; vect->y = e1->y - e2->y; vect->z = e1->z - e2->z; return vect;

}

Elem \*World::vectMultip(Elem \*e1, Elem \*e2)

{

Elem \*vect = new Elem; vect->x = e1->y \* e2->z - e1->z \* e2->y; vect->y = e1->z \* e2->x - e1->x \* e2->z;

vect->z = e1->x \* e2->y - e1->y \* e2->x;

return vect;

}

float World::pointToLineDist(Elem \*e1, Elem \*e2, Elem \*obst) {

Elem \*e3, \*e4, \*e3\_e4\_mult; float mult\_modul, e3\_modul;

e3 = vectDiff(e1, e2); //âåêòîð - ðàçíèöà ìåæäó íà÷àëüíîé òî÷êîé ïðÿìîé è êîíå÷íîé e4 = vectDiff(obst, e2); //âåêòîð - ðàçíèöà ìåæäó òî÷êîé ïðåïÿòñòâèÿ è íà÷àëüíîé e3\_e4\_mult = vectMultip(e3, e4); //âåêòîðíîå ïðîèçâåäåíèå ýòèõ âåêòîðîâ mult\_modul = vectModul(e3\_e4\_mult); // äëèíà ýòîãî âåêòîðà

e3\_modul = vectModul(e3); // äëèíà âåêòîðà ìåæäó íà÷àëüíîé òî÷êîé ïðÿìîé è êîíå÷íîé return mult\_modul / e3\_modul;

}

void World::addEdge(Node \*node1, Node \*node2)

{

++intersections\_count; node1->edges.push\_back(node2); node2->edges.push\_back(node1);

}

void World::generateNodes(int count)

{

nodes.reserve(count);

float zenit, azimut, r;

if (obstacles.size() > 0)

{

int i = 0;

Node \*tmp\_node = new Node;

while (i < count)

{

getCords(zenit, azimut, r);

tmp\_node->setCords(zenit, azimut, r, node\_radius);

if (!isIntersect(obstacles.front(), tmp\_node))

{

Node \*node = new Node(zenit, azimut, r, node\_radius); nodes.push\_back(node);

++i;

}

}

delete tmp\_node;

}

else

{

for (int i = 0; i < count; ++i)

{

getCords(zenit, azimut, r);

Node \*node = new Node(zenit, azimut, r, node\_radius); nodes.push\_back(node);

}

}

}

void World::generateObstacle()

{

srand(time(NULL)); float zenit, azimut, r; getCords(zenit, azimut, r);

Obstacle \*obstacle = new Obstacle(zenit, azimut, r, obst\_radius); obstacles.push\_back(obstacle);

}

bool World::isIntersect(Elem \*node1, Elem \*node2)

{

return (vectDist(node1, node2) < node\_radius);

}

void World::checkEdges()

{

#pragma omp parallel for

for (int i = 0; i < nodes.size(); ++i)

{

for (int j = i+1; j < nodes.size(); ++j)

{

if (isIntersect(nodes[i], nodes[j]))

{

if (!nodes[i]->findNode(nodes[j]))

{

if (obstacles.size() > 0)

{

if (pointToLineDist(nodes[i], nodes[j], obstacles.front()) > obstacles.front()->radius)

if (!nodes[i]->findNode(nodes[j]))

#pragma omp critical

{

addEdge(nodes[i], nodes[j]);

} }

else

#pragma omp critical

{

addEdge(nodes[i], nodes[j]);

}

}

}

}

}

}

Класс World.h:

#include <iostream>

#include <vector>

#include <list>

using namespace std;

class Elem { public:

Elem() {}

Elem(float, float, float, float); float x, y, z; float radius; public:

void printDec();

void setCords(float, float, float, float);

};

class Node : public Elem

{ public:

Node() {}

Node(float, float, float, float); bool findNode(Node \*); list<Node\*> edges;

bool visited;

};

class Obstacle : public Elem

{ public:

Obstacle() {}

Obstacle(float, float, float, float);

};

class World { public:

World(float, float, float);

World();

World(const World &obj);

~World();

void generateNodes(int); void generateObstacle(); void addEdge(Node \*, Node \*);

void clearWorld();

float getWorldRadius() const

{

return world\_radius;

}

float getNodeRadius() const

{

return node\_radius;

}

float getObstacleRadius() const

{

return obst\_radius;

}

int getIntersectionsCount() const

{

return intersections\_count;

}

float getNodesCount() const

{

return nodes\_count;

}

void setWorldRadius(float val)

{

world\_radius = val;

}

void setNodeRadius(float val)

{

node\_radius = val;

}

void setObstalceRadius(float val)

{

obst\_radius = val;

}

void setIntersectionsCount(int val)

{

intersections\_count = val;

}

void setNodesCount(int val)

{

nodes\_count = val;

}

vector<Node\*> getNodes() const

{

return nodes;

}

vector<Obstacle\*> getObstacles() const

{

return obstacles;

}

void checkEdges(); int intersectionsCount() const; bool isIntersect(Elem \*, Elem \*); void DFS(Node \*cur\_node);

int componentsCount();

private:

vector<Node\*> nodes; vector<Obstacle\*> obstacles; private:

float world\_radius; float node\_radius; float obst\_radius; int intersections\_count; int nodes\_count; private:

float pointToLineDist(Elem \*, Elem \*, Elem\*); float vectDist(Elem \*, Elem \*); float vectModul(Elem \*); Elem \*vectDiff(Elem \*, Elem \*);

Elem \*vectMultip(Elem \*, Elem \*);

void getCords(float&, float&, float&);

};

Состав серверного скрипта: Класс Request.php:

<?php

/\*\*

* Created by PhpStorm.
* User: User
* Date: 08.04.2016
* Time: 13:04

\*/

class Request

{

private $iterationsCount; private $nodeRadius; private $obstacleRadius; private $worldRadius; private $nodesCount;

private $hasObstacle;

public function \_\_construct()

{

$this->iterationsCount = $\_GET[ITERATIONS\_COUNT\_PARAM];

$this->nodeRadius = $\_GET[NODE\_RADIUS\_PARAM];

$this->obstacleRadius = $\_GET[OBSTACLE\_RADIUS\_PARAM];

$this->worldRadius = $\_GET[WORLD\_RADIUS\_PARAM]; $this->nodesCount = $\_GET[NODES\_COUNT\_PARAM];

$this->hasObstacle = $\_GET[HAS\_OBSTACLE\_PARAM];

}

public static function parseParams()

{

if (!isset($\_GET[NODES\_COUNT\_PARAM])) {

return false;

}

if (!isset($\_GET[ITERATIONS\_COUNT\_PARAM])) { return false;

}

if (!isset($\_GET[NODE\_RADIUS\_PARAM])) {

return false;

}

if (!isset($\_GET[OBSTACLE\_RADIUS\_PARAM])) { return false;

}

if (!isset($\_GET[WORLD\_RADIUS\_PARAM])) {

return false;

}

if (!isset($\_GET[HAS\_OBSTACLE\_PARAM])) {

return false;

} return true;

}

/\*\*

* @return mixed

\*/

public function getIterationsCount()

{

return $this->iterationsCount;

}

/\*\*

* @return mixed

\*/

public function getNodeRadius()

{

return $this->nodeRadius;

}

/\*\*

* @return mixed

\*/

public function getObstacleRadius()

{

return $this->obstacleRadius;

}

/\*\*

* @return mixed

\*/

public function getWorldRadius()

{

return $this->worldRadius;

}

/\*\*

* @return mixed

\*/

public function getNodesCount()

{

return $this->nodesCount;

}

/\*\*

* @return mixed

\*/

public function getHasObstacle()

{

return $this->hasObstacle;

}

public function toCmdArgs()

{

return $this->iterationsCount . " " . $this->nodeRadius . " "

. $this->obstacleRadius . " "

. $this->worldRadius . " "

. $this->nodesCount . " "

. $this->hasObstacle;

}

}

Класс Result.php:

<?php

/\*\*

* Created by PhpStorm. \* User: User
* Date: 08.04.2016
* Time: 14:13

\*/ class Result { private $nodesAvg; private $dispersion;

private $compsAvg;

public function \_\_construct($cmdOutput)

{

$vars = explode(" ", $cmdOutput);

$this->nodesAvg = $vars[0];

$this->dispersion = $vars[1];

$this->compsAvg = $vars[2];

}

public function echoJson()

{

echo json\_encode(array("status" => "ok", "result"=>array("nodes" => $this->nodesAvg,

"dispersion" => $this->dispersion,

"components" => $this->compsAvg)));

}

/\*\*

* @return mixed

\*/

public function getNodesAvg()

{

return $this->nodesAvg;

}

/\*\*

* @return mixed

\*/

public function getDispersion()

{

return $this->dispersion;

}

/\*\*

* @return mixed

\*/

public function getCompsAvg()

{

return $this->compsAvg;

}

}

Класс DbOperations.php:

<?php

class DbOperations

{

static function findResult($connection, $request)

{

if (!$stmt = $connection->prepare("SELECT nodesAvg,dispersion,compsAvg

FROM results WHERE id =

(SELECT params.id\_result FROM params WHERE params.nodesCount = (?)

AND params.iterationsCount = (?)

AND params.nodeRadius = (?)

AND params.obstacleRadius = (?)

AND params.worldRadius = (?)

AND params.hasObstacle =(?))")

) {

return null;

}

if (!$stmt->bind\_param('iiiiii', $request->getNodesCount(),

$request->getIterationsCount(),

$request->getNodeRadius(),

$request->getObstacleRadius(),

$request->getWorldRadius(),

$request->getHasObstacle())

) {

return null;

}

if (!$stmt->execute()) {

return null;

}

if (!$result = $stmt->get\_result()) {

return null;

}

$object = $result->fetch\_object();

return $object;

}

public static function insertExperiment($connection, $params, $results)

{

$connection->begin\_transaction();

$resultsId = self::insertResult($connection, $results); if ($resultsId == -1) { $connection->rollback();

return false;

} else {

if (!$stmt = $connection->prepare("INSERT INTO params(nodesCount, iterationsCount, nodeRadius, obstacleRadius, worldRadius, hasObstacle

,id\_result) VALUES (?,?,?,?,?,?,?)")

) {

return -1;

}

if (!$stmt->bind\_param('iiiiiii', $params->getNodesCount(),

$params->getIterationsCount(),

$params->getNodeRadius(),

$params->getObstacleRadius(),

$params->getWorldRadius(),

$params->getHasObstacle(),

$resultsId)

) {

return -1;

}

if (!$stmt->execute()) { $connection->rollback();

return false;

}

$connection->commit();

return true;

}

}

private static function insertResult($connection, $result)

{

if (!$stmt = $connection->prepare("INSERT INTO results(nodesAvg, dispersion, compsAvg) VALUES (?,?,?)")

) {

return -1;

}

if (!$stmt->bind\_param('ddd', $result->getNodesAvg(),

$result->getDispersion(),

$result->getCompsAvg())

) {

return -1;

}

if (!$stmt->execute()) {

return -1;

}

$inserted\_user\_id = $connection->insert\_id;

$stmt->close();

return $inserted\_user\_id;

} }

Файл Server.php:

<?php

ini\_set("max\_execution\_time", "120"); include "Request.php"; include "DbOperations.php";

include "Result.php";

define("NODES\_COUNT\_PARAM", "nodesCount"); define("ITERATIONS\_COUNT\_PARAM", "iterationsCount"); define("NODE\_RADIUS\_PARAM", "nodeRadius"); define("OBSTACLE\_RADIUS\_PARAM", "obstacleRadius"); define("WORLD\_RADIUS\_PARAM", "worldRadius"); define("HAS\_OBSTACLE\_PARAM", "hasObstacle");

function connectToDb()

{

$conn = new mysqli('localhost', 'root', '', "course");

$conn->autocommit(false); if ($conn->connect\_errno) {

echo "Не удалось подключиться к MySQL: ("

. $conn->connect\_errno . ") "

. $conn->connect\_error;

}

return $conn;

}

if ($\_SERVER["REQUEST\_METHOD"] == "GET") {

if (!Request::parseParams()) {

die(json\_encode(array("status" => "error", "cause" => "params missed")));

}

$request = new Request();

$connection = connectToDb();

$connection->begin\_transaction();

$findResult = DbOperations::findResult($connection, $request);

if ($findResult == null) { $connection->rollback();

exec("courseParallel.exe " . $request->toCmdArgs(), $output);

$programResult = new Result($output[0]);

$programResult->echoJson();

$insertResult = DbOperations::insertExperiment($connection, $request, $programResult); } else {

$connection->commit();

echo json\_encode(array("status" => "ok", "result" => array("nodes" => $findResult>nodesAvg,

"dispersion" => $findResult->dispersion,

"components" => $findResult->compsAvg)));

} }

Сетевая часть клиентского приложения:

Класс OkHttpHelper.java:

package chuprin.serg.diplomclient.utils; import android.util.Log; import org.json.JSONException; import org.json.JSONObject; import java.io.IOException; import java.util.concurrent.TimeUnit; import chuprin.serg.diplomclient.data.Params; import okhttp3.Call; import okhttp3.Callback; import okhttp3.HttpUrl; import okhttp3.OkHttpClient; import okhttp3.Request;

import okhttp3.Response;

public class OkHttpHelper { public static final String STATUS = "status"; public static final String STATUS\_OK = "ok"; public static final String STATUS\_ERROR = "error"; public static final String JSON\_RESULT = "result"; public static final String JSON\_RESULT\_NODES = "nodes"; public static final String JSON\_RESULT\_COMPS = "components"; public static final String JSON\_RESULT\_DISPER = "dispersion"; public static final String NODES\_COUNT\_PARAM = "nodesCount"; public static final String ITERS\_COUNT\_PARAM = "iterationsCount"; public static final String NODE\_RADIUS\_PARAM = "nodeRadius"; public static final String OBSTACLE\_RADIUS\_PARAM = "obstacleRadius"; public static final String WORLD\_RADIUS\_PARAM = "worldRadius"; public static final String HAS\_OBSTACLE\_PARAM = "hasObstacle";

private static final OkHttpHelper ourInstance = new OkHttpHelper();

@SuppressWarnings("FieldCanBeLocal") private final String mServerUrl = "192.168.1.133"; // private final String mServerUrl = "192.168.43.171"; private final OkHttpClient mHttpClient;

private final Request.Builder builder;

private OkHttpHelper() { mHttpClient = new OkHttpClient.Builder() .connectTimeout(15, TimeUnit.SECONDS)

.readTimeout(2, TimeUnit.MINUTES)

.build();

builder = new Request.Builder();

}

public static OkHttpHelper getInstance() {

return ourInstance;

}

public void getRequest(final Params myParams,

final iServerDbResponseEventListener callback) {

HttpUrl.Builder paramsBuilder = new HttpUrl.Builder()

.scheme("http")

.host(mServerUrl)

.addPathSegment("course")

.addPathSegment("Server.php"); paramsBuilder.addPathSegment(mServerUrl); paramsBuilder.addQueryParameter(NODES\_COUNT\_PARAM, String.valueOf(myParams.getNodesCount()));

paramsBuilder.addQueryParameter(ITERS\_COUNT\_PARAM, String.valueOf(myParams.getItersCount()));

paramsBuilder.addQueryParameter(NODE\_RADIUS\_PARAM, String.valueOf(myParams.getNodeRadius()));

paramsBuilder.addQueryParameter(OBSTACLE\_RADIUS\_PARAM, String.valueOf(myParams.getObstacleRadius()));

paramsBuilder.addQueryParameter(WORLD\_RADIUS\_PARAM, String.valueOf(myParams.getWorldRadius()));

paramsBuilder.addQueryParameter(HAS\_OBSTACLE\_PARAM,

String.valueOf(myParams.getHasObstacle()));

Request request = builder.url(paramsBuilder.build().toString()).method("GET", null).build();

mHttpClient.newCall(request).enqueue(new Callback() {

@Override

public void onFailure(Call call, IOException e) { if (callback != null) {

callback.onFailure();

}

}

@Override

public void onResponse(Call call, Response response) throws IOException {

String responseBody = response.body().string(); Log.i("JsonResponse", responseBody);

if (callback != null) { try {

callback.onResponse(new JSONObject(responseBody)); } catch (JSONException e) { e.printStackTrace();

callback.onFailure();

}

} else {

Log.i("TAG", "okHttpHelper post onResponse: callback null");

}

}

});

}

public interface iServerDbResponseEventListener { void onFailure();

void onResponse(JSONObject response);

}

}

Класс взаимодействия с сервером ServerInteractor.java:

package chuprin.serg.diplomclient.interactors;

import org.json.JSONException;

import org.json.JSONObject;

import chuprin.serg.diplomclient.data.Params; import chuprin.serg.diplomclient.data.Result;

import chuprin.serg.diplomclient.interactors.interfaces.iServerInteractor; import chuprin.serg.diplomclient.utils.OkHttpHelper;

public class ServerInteractorImpl implements iServerInteractor {

@Override

public void getExperimentResult(final Params params, final iOnServerResponseListener listener) {

OkHttpHelper.getInstance().getRequest(params, new

OkHttpHelper.iServerDbResponseEventListener() {

@Override public void onFailure() { if (listener != null) {

listener.onServerRequestFailed();

}

}

@Override

public void onResponse(JSONObject jsonResponse) { if (jsonResponse != null) { try {

if (listener != null) { if

(jsonResponse.getString(OkHttpHelper.STATUS).equals(OkHttpHelper.STATUS\_OK)) { listener.onServerResponse(params,parseResponse(jsonResponse));

} else {

listener.onServerRequestFailed();

}

}

} catch (JSONException e) { e.printStackTrace();

}

}

}

});

}

private Result parseResponse(JSONObject jsonObject) { try {

JSONObject dataSet = jsonObject.getJSONObject(OkHttpHelper.JSON\_RESULT); return new Result(dataSet.getDouble(OkHttpHelper.JSON\_RESULT\_NODES), dataSet.getDouble(OkHttpHelper.JSON\_RESULT\_DISPER), dataSet.getDouble(OkHttpHelper.JSON\_RESULT\_COMPS)); } catch (JSONException e) { e.printStackTrace();

return null;

}

}

}

Класс для работы с базой данных DatabaseInteractor.java:

package chuprin.serg.diplomclient.interactors;

import android.content.ContentValues; import android.content.Context; import android.database.Cursor; import android.database.sqlite.SQLiteDatabase; import android.support.annotation.NonNull;

import chuprin.serg.diplomclient.data.BaseParams; import chuprin.serg.diplomclient.data.Params; import chuprin.serg.diplomclient.data.Result;

import chuprin.serg.diplomclient.interactors.interfaces.iDatabaseInteractor; import chuprin.serg.diplomclient.utils.DatabaseHelper;

public class DatabaseInteractorImpl implements iDatabaseInteractor { private DatabaseHelper mDatabaseHelper;

public DatabaseInteractorImpl(Context context) { mDatabaseHelper = new DatabaseHelper(context);

}

@Override

public Result getResult(Params params) {

SQLiteDatabase sqLiteDatabase = mDatabaseHelper.getReadableDatabase();

Cursor cursor = sqLiteDatabase.rawQuery("SELECT " +

DatabaseHelper.ResultTable.NODES\_AVG + "," +

DatabaseHelper.ResultTable.DISPERSION + "," +

DatabaseHelper.ResultTable.COMPONENTS\_AVG + " FROM " +

DatabaseHelper.ResultTable.TABLE\_NAME + " " +

"WHERE " + DatabaseHelper.ResultTable.\_ID + " = (SELECT " + DatabaseHelper.ParamsTable.ID\_RESULT + " FROM " +

DatabaseHelper.ParamsTable.TABLE\_NAME + " WHERE " +

DatabaseHelper.ParamsTable.NODES\_COUNT + " = ? AND " +

DatabaseHelper.ParamsTable.ITERS\_COUNT + " = ? " + "AND " +

DatabaseHelper.ParamsTable.NODE\_RADIUS + " = ? AND " +

DatabaseHelper.ParamsTable.OBSTACLE\_RADIUS + " = ? " + "AND " +

DatabaseHelper.ParamsTable.WORLD\_RADIUS + " = ? AND " + DatabaseHelper.ParamsTable.HAS\_OBSTACLE + " = ?)", new String[]{

String.valueOf(params.getNodesCount()),

String.valueOf(params.getItersCount()),

String.valueOf(params.getNodeRadius()),

String.valueOf(params.getObstacleRadius()),

String.valueOf(params.getWorldRadius()),

String.valueOf(params.getHasObstacle())}); Result result = null;

if (cursor != null) { if (cursor.moveToFirst()) {

float nodesAvg =

cursor.getFloat(cursor.getColumnIndex(DatabaseHelper.ResultTable.NODES\_AVG));

float dispersion =

cursor.getFloat(cursor.getColumnIndex(DatabaseHelper.ResultTable.DISPERSION));

float compsAvg =

cursor.getFloat(cursor.getColumnIndex(DatabaseHelper.ResultTable.COMPONENTS\_AVG)); result = new Result(nodesAvg, dispersion, compsAvg);

}

cursor.close();

}

return result;

}

@Override

public void insertResult(Params params, Result result) {

long insertedResultId = insertResult(result);

if (insertedResultId != -1) {

SQLiteDatabase database = mDatabaseHelper.getWritableDatabase();

ContentValues values = new ContentValues();

values.put(DatabaseHelper.ParamsTable.HAS\_OBSTACLE, params.getHasObstacle()); values.put(DatabaseHelper.ParamsTable.ID\_RESULT, insertedResultId); values.put(DatabaseHelper.ParamsTable.ITERS\_COUNT, params.getItersCount()); values.put(DatabaseHelper.ParamsTable.NODE\_RADIUS, params.getNodeRadius()); values.put(DatabaseHelper.ParamsTable.NODES\_COUNT, params.getNodesCount()); values.put(DatabaseHelper.ParamsTable.OBSTACLE\_RADIUS, params.getObstacleRadius());

values.put(DatabaseHelper.ParamsTable.WORLD\_RADIUS, params.getWorldRadius()); database.insert(DatabaseHelper.ParamsTable.TABLE\_NAME, null, values);

}

}

@Override

public Cursor getResult(BaseParams params) { if (params.getHasObstacle() == 1) {

return getResultHasObstacle(params);

} else { return getResultNoObstacle(params);

}

}

@NonNull

public Cursor getResultById(int idResult) {

// return

mDatabaseHelper.getReadableDatabase().query(DatabaseHelper.ResultTable.TABLE\_NAME,

// null,

// DatabaseHelper.ResultTable.\_ID + "=?",

// new String[]{String.valueOf(idResult)},

// null,

// null,

// null);

return mDatabaseHelper.getReadableDatabase().rawQuery("SELECT \* FROM "

+ DatabaseHelper.ResultTable.TABLE\_NAME + " WHERE "

+ DatabaseHelper.ResultTable.\_ID + " = " + idResult, null);

}

@Override

public Cursor getAllParams() {

return mDatabaseHelper.getReadableDatabase().rawQuery("SELECT \* FROM " + DatabaseHelper.ParamsTable.TABLE\_NAME, null);

}

private Cursor getResultHasObstacle(BaseParams params) {

SQLiteDatabase sqLiteDatabase = mDatabaseHelper.getReadableDatabase();

String query = "SELECT \* FROM " + DatabaseHelper.ParamsTable.TABLE\_NAME + " " +

"INNER JOIN " + DatabaseHelper.ResultTable.TABLE\_NAME + " ON " +

DatabaseHelper.ParamsTable.TABLE\_NAME + "." +

DatabaseHelper.ParamsTable.ID\_RESULT + " = " +

DatabaseHelper.ResultTable.TABLE\_NAME + "." +

DatabaseHelper.ResultTable.\_ID + " AND " +

DatabaseHelper.ParamsTable.NODE\_RADIUS + " = ? AND " +

DatabaseHelper.ParamsTable.OBSTACLE\_RADIUS + " = ? " + "AND " +

DatabaseHelper.ParamsTable.WORLD\_RADIUS + " = ? AND " +

DatabaseHelper.ParamsTable.HAS\_OBSTACLE + " = ?" +

"ORDER BY " + DatabaseHelper.ParamsTable.NODES\_COUNT + " ASC"; return sqLiteDatabase.rawQuery(query, new String[]{

String.valueOf(params.getNodeRadius()),

String.valueOf(params.getObstacleRadius()),

String.valueOf(params.getWorldRadius()),

String.valueOf(params.getHasObstacle())});

}

private Cursor getResultNoObstacle(BaseParams params) {

SQLiteDatabase sqLiteDatabase = mDatabaseHelper.getReadableDatabase();

String query = "SELECT \* FROM " + DatabaseHelper.ParamsTable.TABLE\_NAME + " " +

"INNER JOIN " + DatabaseHelper.ResultTable.TABLE\_NAME + " ON "

+ DatabaseHelper.ParamsTable.TABLE\_NAME + "." +

DatabaseHelper.ParamsTable.ID\_RESULT + " = " +

DatabaseHelper.ResultTable.TABLE\_NAME + "." +

DatabaseHelper.ResultTable.\_ID + " AND " +

DatabaseHelper.ParamsTable.NODE\_RADIUS + " = ? AND " +

DatabaseHelper.ParamsTable.WORLD\_RADIUS + " = ? " + "AND " +

DatabaseHelper.ParamsTable.HAS\_OBSTACLE + " = ? " +

"ORDER BY " + DatabaseHelper.ParamsTable.NODES\_COUNT + " ASC"; return sqLiteDatabase.rawQuery(query, new String[]{

String.valueOf(params.getNodeRadius()),

String.valueOf(params.getWorldRadius()),

String.valueOf(params.getHasObstacle())});

}

private long insertResult(Result result) {

SQLiteDatabase database = mDatabaseHelper.getWritableDatabase();

ContentValues values = new ContentValues();

values.put(DatabaseHelper.ResultTable.COMPONENTS\_AVG, result.getComponents());

values.put(DatabaseHelper.ResultTable.DISPERSION, result.getDispersion()); values.put(DatabaseHelper.ResultTable.NODES\_AVG, result.getNodes()); return database.insert(DatabaseHelper.ResultTable.TABLE\_NAME, null, values);

}