**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

# Факультет физико-математических и естественных наук Кафедра прикладной информатики и теории вероятностей

**ОТЧЕТ**

**ПО УЧЕБНОЙ ПРАКТИКЕ**

Студент: Косолапов Степан Эдуардович

Группа: НПИбд-01-20

# МОСКВА

2023 г.

# Постановка задачи

Установить и изучить среду для моделирования сетей VANET – SUMO. Установить ns3 и приобрести навыки работы с ns3.

# Выполнение работы

# Установка SUMO

# Выполнять установку будем на операционной системе macOS.

# Установим sumo через менеджера пакетов brew. Для этого переходим на официальный сайт SUMO и проходим по указанным шагам: <https://sumo.dlr.de/docs/Installing/index.html#macos>.

# Обновляем brew: brew update

# Устанавливаем XQuartz: brew install --cask xquartz

# Устанавливаем SUMO: brew tap dlr-ts/sumo brew install sumo

# Добавляем в файл конфигурации shell(мы используем zsh) переменную окружения SUMO\_HOME: Проверяем, что переменная добавилась: Text Description automatically generated

# Далее установим графический интерфейс для sumo – sumo-gui. Делаем это так же через менеджера пакетов brew: brew install --cask sumo-gui

# Теперь надо перезапустить операционную систему.

# После этого нам доступно приложение sumo-gui. Открываем его: Graphical user interface, text, application, Word Description automatically generated

# Симуляция в SUMO

# Для ознакомления попробуем запустить какую-нибудь симуляцию. Есть много способов создать файл симуляции, но самый быстрый для нас сейчас – это использовать [Web Wizard](https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html). Это программа, позволяющая нам создать файл симуляции через графический интерфейс. При этом мы можем просимулировать практически любое место на карте. Web Wizard использует [openstreetmap](https://www.openstreetmap.org/) для создания симуляции.

# Чтобы запустить Web Wizard – откроем официальную документацию и пройдем по шагам: <https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html>

# Нам нужно запустить файл osmWebWizard.py. Он лежит в директории tools. Graphical user interface, application Description automatically generated Запускаем этот файл через интерпретатор python: Map Description automatically generated

# Открывается графический интерфейс. Здесь мы можем выбрать интересующие нас параметры. Например, город и количество автомобилей на дорогах: Graphical user interface, application Description automatically generated Graphical user interface, text, application Description automatically generated

# Нажимаем generate scenario и файл со сгенерированной сетью открывается в sumo: Graphical user interface, diagram Description automatically generated

# В консоли снизу мы видим откуда был открыт файл. Text Description automatically generated

# Для интереса мы можем его изучить: Мы видим, что это файл xml. В его начале есть какая-то конфигурация: Text Description automatically generated Далее идет само описание сгенерированной сети: Text Description automatically generated Из понятного – мы видим тут ребра, которые отображаются на карте. Так же можно разобрать, что на нашей карте будут перекрёстки – они описаны в тэгах junction Text Description automatically generated

# Попробуем запустить теперь эту симуляцию. Для этого нажимаем кнопку Run в SUMO: Graphical user interface, text Description automatically generated

# Мы сразу увидим, как по городу начнут передвигаться желтые автомобили: Graphical user interface Description automatically generated with medium confidence

# Мы можем уменьшить скорость их передвижения с помощью изменения параметра Delay в большую сторону:

# Мы так же можем выполнять симуляцию пошагово кнопкой Step: Graphical user interface Description automatically generated

# Экспорт трейса из SUMO

# В этапе 2 мы рассматривали создание файла .sumocfg через WebWizard. Это файл конфигурации, который описывает сценарий нашей симуляции. Мы можем прогнать этот сценарий через sumo и на выходе получить файл, который будет описывать состояние системы в каждый промежуток времени. Чтобы узнать, как это сделать, мы можем обратиться к официальной документации - <https://sumo.dlr.de/docs/Tutorials/Trace_File_Generation.html>.

# Пройдем по шагам, указанным в документации: Запускаем sumo, передавая на вход файл нашего сценария. В качестве output указываем, что нам нужен fcd файл, то есть floating car data файл(см [документацию](https://sumo.dlr.de/docs/Simulation/Output/FCDOutput.html)), и название выходного файла, назовем его sumoTrace.xml: Text Description automatically generated with medium confidence

# Видим логи запуска симуляции: Тут есть кое-какая статистика по итогам симуляции. Из интересного, видим, что в программе, видимо, был использован алгоритм Дейкстры для поиска кратчайшего пути. В документации мы можем найти подтверждение этой догадке - <https://sumo.dlr.de/docs/Simulation/Routing.html#routing_algorithms> В документации так же говорится, что мы даже можем выбрать astar вместо алгоритма Дейкстры, если нам это нужно, просто указав опцию --routing-algorithm Text, letter Description automatically generated

# После выполнения сценария мы получаем на выходе xml файл с трейсом:

# Взглянем на получившийся файл: В нем мы видим набор тэгов timestep, которые содержат в себе состояние системы в определенный промежуток времени. Текущий промежуток времени определяется аттрибутом time. В нашем случае – внутри timestep присутствуют тэги, обозначающие состояние автомобиля. Каждый автомобиль имеет свой уникальный идентификатор, координаты, тип, скорость и другие параметры.

# Graphical user interface, text, application Description automatically generated Можно заметить, что с течением времени старые автомобили пропадают(выезжают за карту), а новые автомобили добавляются: Text Description automatically generated

# Установка ns-3

# Теперь нам нужно установить ns-3. Для начала скачаем с [официального сайта](https://www.nsnam.org/releases/ns-3-38/download/) архив с кодом: Graphical user interface, text, application Description automatically generatedРаспаковываем архив и видим ряд файлов, Text Description automatically generated

# Теперь мы можем запустить файл build.py: Стоит заметить, что предварительно требуется установить clang и gcc, cmake и python. Началась компиляция файлов: Text Description automatically generated

# После установки заходим в директорию ns-3.38.

# В ней лежат все скомпилированные файлы, они готовы к использованию:

# Мы можем провалидировать установку, запустив команду ./test.py: Text Description automatically generated

# Все скомпилировалось успешно:

# Теперь мы можем запустить скомпилированный ns3. Так как мы указали –enable-examples при компиляции, то у нас есть ряд примеров, которые мы можем запустить. Например, запустим hello-simulator: Graphical user interface Description automatically generated with low confidence

# Видим, что все корректно работает.

# Преобразование трейса в файл .tcl

# Теперь, когда у нас есть трейс, мы можем преобразовать его в тот формат, который мы можем использовать в ns3. Например .tcl. Чтобы узнать как это сделать – смотрим официальную документацию sumo - <https://sumo.dlr.de/docs/Tools/TraceExporter.html#ns2ns3>

# Пройдем по шагам из документации. Нужно запустить скрипт traceExporter.py из папки tools: Text Description automatically generated

# После выполнения скрипта – видим наш .tcl файл:

# Открыв этот файл, мы видим список нод: Тут после at указан timestep. Для каждого timestep указана строка, применяющая позицию(команда setdest) к каждому узлу. Как мы и ожидали, у нас столько же timestep, как и в трейсе, а так же мы видим, что последний узел – 711, а значит общее количество узлов 712, как было в трейсе. То есть файл сконвертировался корректно: Graphical user interface, text Description automatically generated

# Теперь мы можем использовать этот файл в программе для ns3

# Использование файла ns-2 в ns-3

# В поисках каких-то скриптов, позволяющих использовать файл для ns-2 в ns-3, мы нашли пример такого скрипта в официальной документации ns-3: <https://www.nsnam.org/docs/release/3.19/doxygen/ns2-mobility-trace_8cc_source.html>

# В документации видим, что этот скрипт располагается в папке src/mobility/examples: Посмотрим содержимое этого файла: Text Description automatically generated Видим описание того, для чего нужен этот скрипт, как он работает и как его использовать. На вход нужно передать количество узлов, файл трейса(файл для ns-2) и длительность симуляции.

# Попробуем запустить этот скрипт. Для этого указываем путь к tcl файлу, указываем количество узлов в трейс-файле, длительность симуляции. Оставляем неизменным опцию logFile, нам это не важно сейчас.

# Запустилась симуляция и мы видим логи процесса: Text Description automatically generated with medium confidence

# Симуляция прошла успешно, мы смогли использовать файл для ns-2 в ns-3.

# Установка NetAnim

# Для начала мы установим netanim. Мы уже скачали все файлы исходного кода на c++ вместе с пакетом ns-allinone-3.38. Нам нужно собрать из них исполняемый файл(скомпилировать программу)

# Видим, что в папке netanim вместе с кодом у нас лежит файл NetAnim.pro. Можем его использовать для сборки, передав его в программу qmake. Text Description automatically generated

# Эта программа идет вместе с набором графических инструментов QT. Она должна сгенерировать нам makefile, который уже и будет запускать компилятор на нужные файлы и в итоге создаст нам исполняемый файл.

# После первой неудачной попытки произвести сборку make файла, на вывод мы получили множество ошибок. После поисков в интернете, на каком-то форуме удалось найти возможную причину ошибок: Тут говорится, что возможная причина – использование qt6 вместо qt5, т. к. тестирование netanim не проводили на qt6.

# Неудачная попытка сборки привела нас к прочтению readme файла в папке netanim. В нем четко прописывается на каких версиях проводилось тестирование и каким версиям qt поддержка не гарантируется: Graphical user interface, text, application, letter, email Description automatically generated

# Изучив требования по зависимостям, нам нужно установить необходимую версию qt через менеджера пакетов командой brew install qt5.

# После этого мы можем использовать qmake. Передадим на вход в qmake файл NetAnim.pro Text Description automatically generated Видим, что у нас создался makefile.

# Осталось запустить make: A page of a book with writing Description automatically generated with low confidence Видим, что запускается g++ - компилятор, и указываются файлы, которые нужно скомпилировать.

# По итогу видим, что в папке netanim у нас появился исполняемый фал NetAnim: Table Description automatically generated

# Попробуем его запустить: Graphical user interface, text, application Description automatically generated В итоге мы видим графический интерфейс netanim. Так же видим надпись, которая предлагает нам загрузить некий .xml файл трейса. Text Description automatically generated На этом этапе мы имеем только .tcl файл для ns2. Теперь нам нужно как-то преобразовать его в файл трейса через ns3.

# Работа с NetAnim

# На этом этапе мы имеем только .tcl файл для ns2. Теперь нам нужно как-то преобразовать его в файл трейса через ns3.

# Наша задача получить файл для воспроизведения в NetAnim. Для этого нужно поискать информацию в документации. Находим указание, как это можно сделать: <https://www.nsnam.org/docs/models/html/animation.html#mandatory> Указания, которые даются в документации: - Ensure that your program's CMakeLists.txt includes the "netanim" module. An example of such a CMakeLists.txt is at src/netanim/examples/CMakeLists.txt.

# - Include the header [#include "ns3/netanim-module.h"] in your test program

# - Add the statement: AnimationInterface anim("animation.xml"); // where "animation.xml" is any arbitrary filename

# Дополним скрипт ns2-mobility-trace. Для начала переместим скрипт в папку scratch. Так нам будет проще добавлять какие-либо заголовочные файлы и в целом это правильный способ писать программы в ns-3.

# Пройдем по шагам из документации: Шаг 1 – убедиться, что в CMakeLists.txt включен модуль netanim. Так как мы будем компилировать файл, находящийся в директории scratch, то этот шаг нам выполнять не нужно. В CMakeLists все прописано таким образом, чтобы нам не пришлось в ручную туда что-то добавлять. Шаг 2 – включить заголовочный файл ns3/netanim-module.h в нашу программу: Text Description automatically generated Добавили заголовочный файл. Шаг 3 – добавить указанное выражение в код программы: Text Description automatically generated Добавили в код программы строчку с этим выражением и указали нужное название файла с трейсом.

# Теперь попробуем заново запустить этот скрипт, уже с отредактированным содержимым: Видим, как сперва файл компилируется. Затем он запускается, и мы видим те же самые логи, как и при предыдущем запуске скрипта: Text Description automatically generated

# Однако, после этого запуска, мы получили на выходе xml файл трейса: A picture containing text Description automatically generated

# Попробуем загрузить этот файл в NetAnim, для этого нажимаем кнопку в левом верхнем углу окна интерфейса NetAnim: Graphical user interface, application, PowerPoint Description automatically generated И выбираем наш xml файл: Graphical user interface, text, application Description automatically generated

# Получаем вот такую картинку: Chart, scatter chart Description automatically generated

# Попробуем увеличить изображение, для этого нажмем на значок приближения: Graphical user interface, application Description automatically generated Так же можно увеличить размер нод в соответствующем поле интерфейса: Graphical user interface, text, application, chat or text message Description automatically generated Итого, мы теперь можем видеть отдельные узлы сети, а также примерно узнать в их расположении изначальную территорию, на которой мы проводили симуляцию в sumo: Chart, scatter chart Description automatically generated

# Попробуем запустить симуляцию, нажав зеленый значок play в интерфейсе программы: Graphical user interface, application Description automatically generated with medium confidence Симуляция запустилась и некоторые точки на канвасе начали двигаться. Это автомобили. Остальные узлы – статичные. Chart, scatter chart Description automatically generated

# Моделирование VANET c помощью ns-3

# Теперь, когда мы научились использовать трейсфайлы ns-2 в программах ns-3, научились создавать эти трейс файлы с помощью sumo и osmWebWizard, нам нужно понять – как моделировать VANET. То, что мы имеем на данном этапе – это просто сценарий передвижения каких-то узлов некой сети с определенной скоростью и по определённой траектории. Чтобы добавить обмен какими-либо сообщениями между узлами сети, нам нужно в программе ns-3 явно это прописать – какой узел в какое время будет отправлять какой пакет, по какому протоколу и тд. Но, для первой попытки моделирования VANET – мы можем воспользоваться примером из документации ns-3, благо она довольно обширная и подробная.

# В поисках основы программы для моделирования VANET на официальном сайте ns-3, мы нашли файл, похожий на то, что требуется нам: <https://www.nsnam.org/docs/release/3.23/doxygen/vanet-routing-compare_8cc_source.html> Судя по названию – это что-то связано с vanet. Изучим поподробнее описание данного файла: A close-up of a computer code Description automatically generated with low confidence Сразу после копирайта, мы видим краткое описание того, что происходит в этом файле. Тут говорится, что эта программа позволяет запустить сценарий симуляции vanet для изучения неких характеристик сети. Важным тут является, что этот скрипт подразумевает использование ns-2 трейсфайлов, а значит мы сможем добавить сюда использование нашего трейсфайла.

# Для начала скопируем этот файл в папку scratch, для лёгкости компиляции и запуска:

# Теперь нам нужно понять, что нужно менять в этом файле, чтобы запустить в нем уже готовый наш трейсфайл. Пробежав по коду – находим метод SetupScenario, что, кажется, похоже на то, что нам нужно: A picture containing text, screenshot, font, line Description automatically generated Посмотрим его содержимое. A screenshot of a computer program Description automatically generated with medium confidence На первый взгляд – все, что тут происходит – это задаются параметры для выполнения сценария. Это так же отражено в названии метода, мы этого ожидали. Нам важно, что тут, в переменной m\_traceFile указывается путь к какому-то трейсфайлу. Возможно это то, что нам нужно, мы попробуем добавить сюда путь к нашему трейсу, но сперва посмотрим на другие параметры. m\_logFile – судя по названию это файл для записи логов. m\_mobility – это нам не понятно. Пробуем загрепать по коду. Находим описание вообще всех переменных, которые задаются в методе. Из этого описания понимаем, что mobility это откуда мы берем трейс файл. 1 – из готового трейса, а 2 – RandomWaypointMobilityModel, то есть случайно сгенерированный трейс. Так же дополнительно изучаем информацию с официальной документации: <https://www.nsnam.org/docs/models/html/wave.html> m\_nNodes – это количество узлов, мы с этим знакомы. m\_TotalSimTime – время симуляции. m\_nodeSpeed и m\_nodePause – это какие-то параметры для узлов. Пока попробуем это не использовать, кажется это не сильно важно, у нас уже есть все параметры перемещения в трейсфайле. Ну и последние два параметра содержат пути для записи csv файлов, в которых будет содержаться какая-то информация по итогам симуляции.

# Проанализировав параметры – попробуем их поменять исходя из того, что у нас в трейс файле: A screenshot of a computer Description automatically generated with medium confidence Мы чуть уменьшили время симуляции просто ради быстроты процесса симуляции и воспроизведения анимации.

# Наивно попробуем положиться на то, что этого нам хватит, чтобы уже иметь какую-то симуляцию vanet на наших узлах и попробуем запустить сценарий.

# Как и в предыдущей нашей попытке запустить ns-2 трейсфайл в ns-3, мы добавляли в скрипт некоторые выражения для NetAnim. Проделаем тут то же самое. Для начала найдем, где же запускается симуляция. Загрепав по слову Run, мы находим нужный метод: A screenshot of a computer program Description automatically generated with medium confidence Расположим тут инициализацию для создания анимации: A screenshot of a computer program Description automatically generated with low confidence Не забываем подключить заголовочный файл:

# Пробуем запускать: A picture containing text, screenshot, font Description automatically generated

# Из-за очень долгого процесса симуляции, было решено немного уменьшить сеть, за счет меньшего количества узлов и длительности симуляции в sumo.

# На обновленном трейсе, состоящем из 46 узлов – симуляция прошла быстро. Видим результирующие файлы: A picture containing text, font, screenshot Description automatically generated

# Попробуем запустить анимацию в NetAnim. A screenshot of a computer screen Description automatically generated with low confidence Видим наши узлы, но пока никак не видим пакеты, которые отправляются.

# Для этого добавим в нашу программу

# выа

# Вывод

Мы изучили графический интерфейс среды для симуляции SUMO. А также попробовали запустить простую симуляцию. Установили ns3.