МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Российский университет дружбы народов имени Патриса Лумумбы»

Факультет физико-математических и естественных наук

Кафедра теории вероятностей и кибербезопасности

«Допустить к защите»

Заведующий кафедрой теории вероятностей и кибербезопасности

д.т.н., профессор

_____ К.Е. Самуйлов

“____”_____20____г.

Выпускная квалификационная работа бакалавра

Направление 09.03.03 «Прикладная информатика»

ТЕМА «Имитационное моделирование самоорганизующихся сетей»

Выполнил студент: С. Э. Косолапов

Группа НПИбд-01-20

Студенческий билет № 1032201745

Руководитель выпускной квалификационной работы:

Королькова А. В., к.ф.-м.н., доцент, доцент кафедры теории вероятностей и кибербезопасности

_____

“____”_____20____г.

Автор: _____

“____”_____20____г.

г. Москва

2024 г.

Федеральное государственное автономное образовательное учреждение высшего образования
«Российский университет дружбы народов имени Патриса Лумумбы»

АННОТАЦИЯ
выпускной квалификационной работы

С. Э. Косолапов

на тему: Имитационное моделирование самоорганизующихся сетей

Выпускная квалификационная работа посвящена имитационному моделированию сетей VANET с использованием инструментов моделирования городской мобильности SUMO и сетевого симулятора NS-2. Целью работы является сравнение протоколов маршрутизации AODV, DSDV и DSR в подвижной автомобильной самоорганизующейся сети со стационарными узлами. А так же разработка необходимого программного комплекса для удобного проведения имитационного моделирования сетей VANET в различных условиях.

В результате работы было получено готовое окружение, подходящее для запуска имитационных экспериментов в сетях VANET. Моделирование мобильности было проведено с помощью симулятора SUMO и Open Street Map, а сетевые взаимодействия производились с помощью программы для NS-2. С помощью Makefile была удобным образом организована интеграция этих инструментов между собой на разных этапах моделирования. Визуализация результата производилась с помощью программы на языке Python и библиотеки matplotlib. Полученные графики были проанализированы и сделан вывод о лучшем протоколе маршрутизации для сетей VANET в описанных условиях.

Автор ВКР _____ С. Э. Косолапов

Оглавление

Список используемых сокращений

Англоязычные сокращения

VANET — Vehicular Ad-hoc Networks, автомобильные самоорганизующиеся сети
NRL — Normalized Routing Load, нормализованная нагрузка маршрутизацией
AODV — Ad hoc On-Demand Distance Vector
DSDV — Destination-Sequenced Distance Vector
DSR — Dynamic Source Routing
TCP — Transmission Control Protocol
UDP — User Datagram Protocol
V2I — Vehicles To Infrastructure
V2V — Vehicles To Vehicles
CAN — Control Area Network
GPS — Global Positioning System
LCS — Local Camera Sensor
OBU — On-board Unit
RSU — Road State Unit
SUMO — Simulation of Urban MObility, название программы
OSM — Open Street Map, название программы
NRL — Normalized Routing Protocol

Русскоязычные сокращения

ИТС — Интеллектуальные транспортные системы

Введение

Работа посвящена имитационному моделированию сетей VANET с использованием инструментов моделирования SUMO и NS-2. Объектом исследования были выбраны протоколы маршрутизации, использующиеся в сетях VANET: DSDV, AODV и DSR. Предметом исследования является сравнение эффективноси данных трёх протоколов маршрутизации в определённых условиях, когда стационарные узлы, находящиеся вне зоны доступа друг от друга, пытаются отправлять данные, используя лишь подвижные узлы, образующие самоорганизующую автомобильную сеть. В рамках работы было проведено подробное описание имитационной модели, а так же был разработан программный комплекс для удобного проведения экспериментов с сетями VANET в различных условиях. Был проведён анализ полученных графиков и сделан вывод об эффективности протоколов маршрутизации сетей VANET в описанных условиях.

Актуальность

Изучаемая тема актуальна по причине развития интеллектуальных транспортных систем в мире и России, а так же из-за небольшого количества актуальных русскоязычных материалов по имитационному моделированию сетей VANET.

Цель работы

Целью выпускной квалификационной работы является написание программного комплекса, необходимого для проведения имитационного моделирования сетей VANET в NS-2 и SUMO. Проведение эксперимента и сравнение трёх различных протоколов маршрутизации в определённых условиях по параметрам эффективности сети.

Задачи

Основными задачами работы являются:

1. Описание имитационной модели.
2. Разработка программного комплекса, необходимого для проведения эксперимента.
3. Анализ полученных результатов и сравнение протоколов маршрутизации.

Структура работы

Работа состоит из введения, трех разделов, заключения и списка используемой литературы. Во введении приведено краткое описание работы, обусловлена актуальность работы, а также поставлена цель и сформулированы задачи выпускной квалификационной работы.

В первом разделе работы дано введение в интеллектуальные транспортные системы и описание особенностей VANET, как части интеллектуальных транспортных систем. Так же рассмотрено устройство протоколов маршрутизации, используемых в сетях VANET, их отличительные особенности.

Во втором разделе работы представлено описание имитационной модели, а так же описаны методологии, используемые для анализа результатов моделирования. Рассмотрены подробнее инструменты моделирования SUMO и NS-2, их особенности и отличия от аналогов.

В третьем разделе выпускной квалификационной работы рассмотрена структура итогового программного комплекса, подробно описаны элементы окружения эксперимента и части программного кода, необходимого для запуска эксперимента. Анализ результатов со сравнением протоколов маршрутизации так же представлен в третьем разделе работы.

В заключении подведены общие итоги работы, изложены основные выводы.

Глава 1

Введение в ИТС. Сети VANET

1.1 Интеллектуальные транспортные системы

Определение и классификация ИТС

Интеллектуальные транспортные системы (ИТС) [1], будучи относительно молодой областью исследований, претерпевают трансформации с новыми идеями и инновациями с быстрыми темпами. Умные автомобили, несмотря на то что являются чрезвычайно важной составляющей ИТС, не являются единственным элементом этих систем, и существуют иные компоненты ИТС, которые получили относительно меньше внимания.

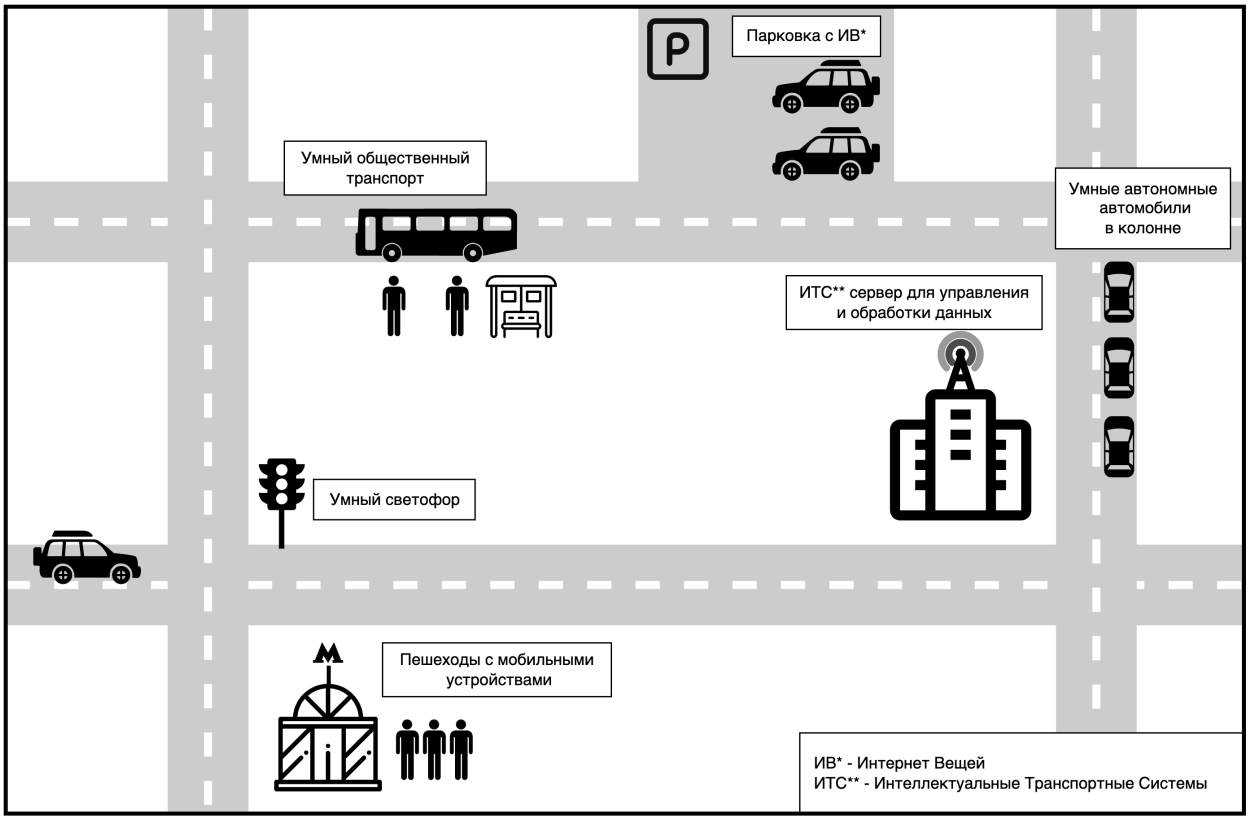

Рис. 1.1: Интеллектуальные транспортные системы

Умные автомобили

Интеллектуальные транспортные системы (ИТС) часто ассоциируют с умными транспортными средствами, которые могут быть оснащены системами помощи водителю, быть полуавтономными или даже полностью автономными. Умные автомобили являются ключевым элементом ИТС из-за значительного количества личных автомобилей на дорогах. По мере развития транспортной отрасли инновации в области умных автомобилей оказывают наибольшее влияние на способы перемещения людей и базовую инфраструктуру транспортных систем. Как показано на рис. 1.1, умные автомобили могут формировать «колонны» с использованием технологии общения между транспортными средствами (V2V), чтобы повысить эффективность передвижения. Данная технология общения и группировки автомобилей стала возможной благодаря исследованиям в области спонтанных сетей для транспортных средств (VANET). VANET представляет собой сеть связи, организованную с использованием беспроводных устройств связи между транспортными средствами, что позволяет обмениваться данными, такими как информация об экстренных ситуациях, расстояние между автомобилями и так далее, для повышения безопасности и эффективности транспортных систем. Система VANET будет рассмотрена позже в данной работе.

Помимо взаимодействия умных автомобилей друг с другом, необходимо также обеспечить коммуникацию между компонентами внутри транспортного средства, такими как электронные управляющие блоки, для реализации различных распределенных функций управления. Это внутреннее общение обеспечивается через сети внутри автомобиля, такие как сеть управления (CAN), CAN с повышенной скоростью передачи данных (CAN FD) и FlexRay.

Общественный транспорт

На сегодняшний день системы общественного транспорта во многих городах являются основным средством передвижения. Маршруты железнодорожного и автобусного транспорта функционируют практически круглосуточно, обеспечивая эффективный и экономичный доступ к перемещению для горожан. Интеллектуальные транспортные системы обладают потенциалом для повышения эффективности и пропускной способности систем общественного транспорта. Умные автобусные остановки могут предоставлять ожидающим пассажирам информацию о расписании автобусов (время прибытия и отправления) и о задержках, как это показано на рис. 1.1. Оптимизация маршрутов с учетом реальных дорожных условий (например, пробок, вызванных другими транспортными средствами и авариями), может повысить комфорт пассажиров и сократить время в пути. Кроме того, системы автобусов и железных дорог также могут извлекать выгоду из взаимодействия с устройствами "Интернета вещей"(ИВ) в рамках интеллек-

туальных транспортных систем, которые передают информацию, такую как количество пассажиров, ожидающих на остановке, или пункт назначения.

Интернет вещей

Смартфоны, являясь примером устройств интернета вещей, играют важную роль в ИТС, обеспечивая не только интеграцию со смарт-автомобилями (например, в целях информационно-развлекательных систем), но и подключение к другим критически важным элементам ИТС, таким как пешеходы. Во всех крупных городах пешеходы используют дорожные пути, такие как переходы, мосты и т.д. Пешеходы могут представлять значительную опасность для транспортных средств и наоборот, поэтому алгоритмы управления светофорами и маршрутизации также должны учитывать пешеходный трафик. С помощью мобильных устройств умные контроллеры движения могут быть уведомлены о пешеходах, ожидающих перехода через улицу. Аналогичным образом, благодаря мобильным устройствам, системы общественного транспорта могут быть уведомлены о пешеходах, ожидающих посадки в метро или автобус. Пока пешеходы ожидают и пользуются системами общественного транспорта, они могут получать обновления о погоде, движении, опасностях, чрезвычайных событиях и т.д. от сети датчиков и сигналов, разбросанных по всей ИТС. Устройства сенсоров и микроконтроллеры являются примерами устройств Интернета вещей, которые все чаще используются в ИТС для различных задач сенсорики и вычислений. Например, эти устройства интернета вещей используются для сбора и обработки ценных аналитических данных для использования в алгоритмах движения и маршрутизации в умных контроллерах движения. Как показано на рис. 1.1, устройства ИВ все чаще используются в различных аспектах ИТС, начиная от систем общественного транспорта и заканчивая парковочными знаками, которые передают информацию о наличии мест путешественникам, ищущим место для парковки. Хотя эти устройства ИВ просты и выполняют свои непосредственные задачи очень экономично и эффективно, они испытывают трудности с реализацией базовых стандартов безопасности и конфиденциальности из-за ограничений ресурсов.

1.2 Сети VANET

Концепция VANET

Транспортные самоорганизующиеся сети (VANET) [2] представляют собой подкатегорию мобильных самоорганизующихся сетей (MANET), обеспечивающих коммуникационную связь между близко расположенными транспортными средствами, а также между транспортными средствами и элементами инфраструктуры. Основой VANET являются придорожные устройства (RSU) и оборудованные транспортные средства с бортовыми устройствами (OBU). Бортовые устройства являются электронными компонентами, установленными на транспортных средствах для обеспечения взаимодействия с придорожными устройствами и другими транспортными средствами, оборудованными аналогичными устройствами. В каждом владеющем OBU транспортном средстве происходит прием сообщений от источника (другого транспортного средства или датчика), их проверка и последующая передача другим участникам дорожного движения через систему связи короткого диапазона (DSRC).

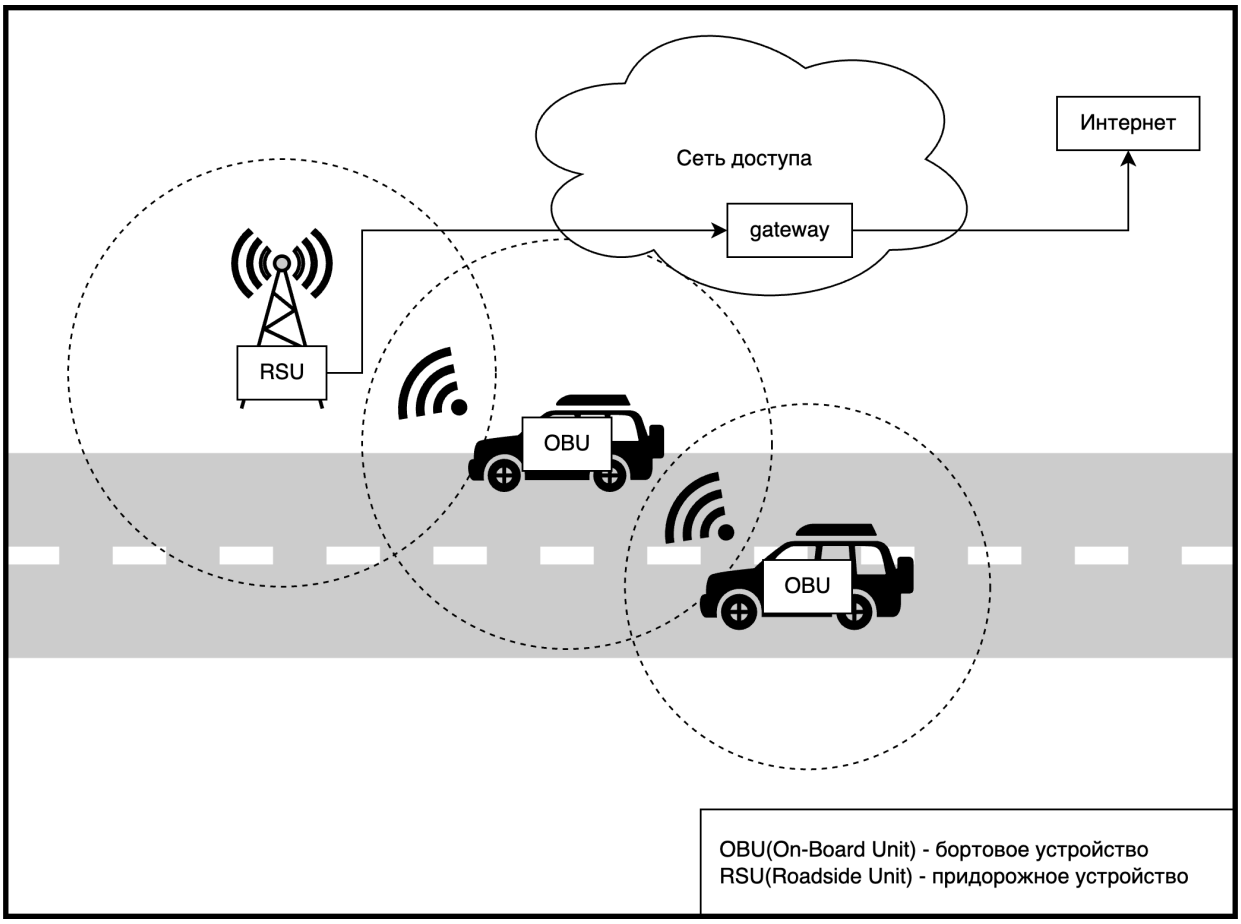

Рис. 1.2: Пример взаимодействия OBU и RSU

Основные компоненты интеллектуальных автомобилей

В современных автомобилях интегрированы различные передовые технологии, предоставляющие услуги для мониторинга в реальном времени и обеспечения безопасности вождения. Эти устройства формируют сенсорную сеть, способную к автономной коммуникации для обмена данными. Информация о внешней среде автомобиля собирается с помощью внутренних датчиков, которые фиксируют и сохраняют данные, такие как параметры сенсоров, навигационные данные, температуру, поведение водителя и изображения для последующего анализа. На основе этой информации автомобиль способен к автономному управлению, которое включает анализ окружающей среды, централизованное принятие решений и выполнение механических действий. Для того чтобы помочь водителям избегать дорожно-транспортных происшествий, необходимо разработать соответствующую системную архитектуру. «Умные автомобили» обычно оснащаются мультирежимными лидарами, микроволновыми радарами, камерами высокого разрешения и т.д., чтобы обеспечить точный и всесторонний сбор данных об окружающей среде. Ключевые компоненты и технологии интеллектуального автомобиля могут включать следующие элементы:

1. Центральный процессор: осуществляет быструю обработку данных, выполняя арифметические, логические операции и операции ввода-вывода.

2. Беспроводной приемопередатчик: обеспечивает передачу данных между автомобилями и между автомобилями и инфраструктурой.

3. GPS-приемник: принимает данные от Глобальной системы позиционирования (GPS), обеспечивая навигационные услуги и точное определение местоположения транспортного средства.

4. Датчики: размещены внутри и снаружи автомобиля для измерения скорости, расстояния до других транспортных средств и других параметров.

5. Интерфейс ввода-вывода: обеспечивает удобную связь между человеком и автомобилем.

6. Радар: использует радиоволны для определения расстояния и отслеживания положения близлежащих транспортных средств.

7. ЛИДАР: датчик, использующий лазерные лучи для точного определения расстояния до объектов.

8. OBU (On-Board Unit): внутренний блок, обеспечивающий подключение автомобиля к различным сетям и устройствам.

9. LCS (Local Camera Sensor): датчик местного наблюдения, отслеживающий действия водителя и точно идентифицирующий объекты вокруг.

Коммуникации в VANET

Разработка VANET становится всё более важной на фоне модернизации общественного транспорта по всему миру, как в бизнесе, так и в личных целях. В режиме реального времени передаваемая информация о состоянии дорожного покрытия или данные, касающиеся безопасности, могут транслироваться через беспроводную сеть между автомобилями или посредством других каналов, включая придорожные устройства, беспилотные летательные аппараты и т.д., способствуя предотвращению дорожных заторов и аварий. Более того, пользователи VANET могут обмениваться развлекательным контентом, вроде новостей, игр, и получать доступ в интернет, что делает длительные поездки более приятными. В результате, VANET способствует улучшению качества вождения, снижая при этом число аварий и заторов.

Для интеграции в сеть VANET транспортные средства должны быть оснащены датчиками, системами навигации, такими как GPS, мультимедийными устройствами и беспроводными модулями. Датчики и мультимедийные технологии могут применяться для обнаружения и идентификации окружающих предметов, включая другие транспортные средства, барьеры и пешеходов, что помогает избегать аварий и неожиданных сбоев. В то время как беспроводные модули расширяют возможности коммуникации, позволяя классифицировать их в зависимости от субъектов связи.

Существует несколько типов коммуникаций в VANET, включая:

1. Прямую связь между транспортными средствами (V2V) без необходимости инфраструктуры.
2. Связь между транспортными средствами и придорожной инфраструктурой (V2I/V2R), например с придорожными устройствами или базовыми станциями.
3. Соединения между различными элементами инфраструктуры (I2I).
4. Связь между транспортными средствами и пешеходами (V2P).
5. Взаимодействие между транспортными средствами и дорожными барьерами (V2B).
6. Соединения между RSU и облачными серверами (V2C).
7. Коммуникация между транспортными средствами и беспилотными летательными аппаратами (V2U).
8. Связь между автомобилем и датчиками (V2S) для передачи информации о дороге.

VANET представляет собой обширную беспроводную сеть с динамичной топологией из-за высокой мобильности транспортных средств и разнообразия маршрутов. Для

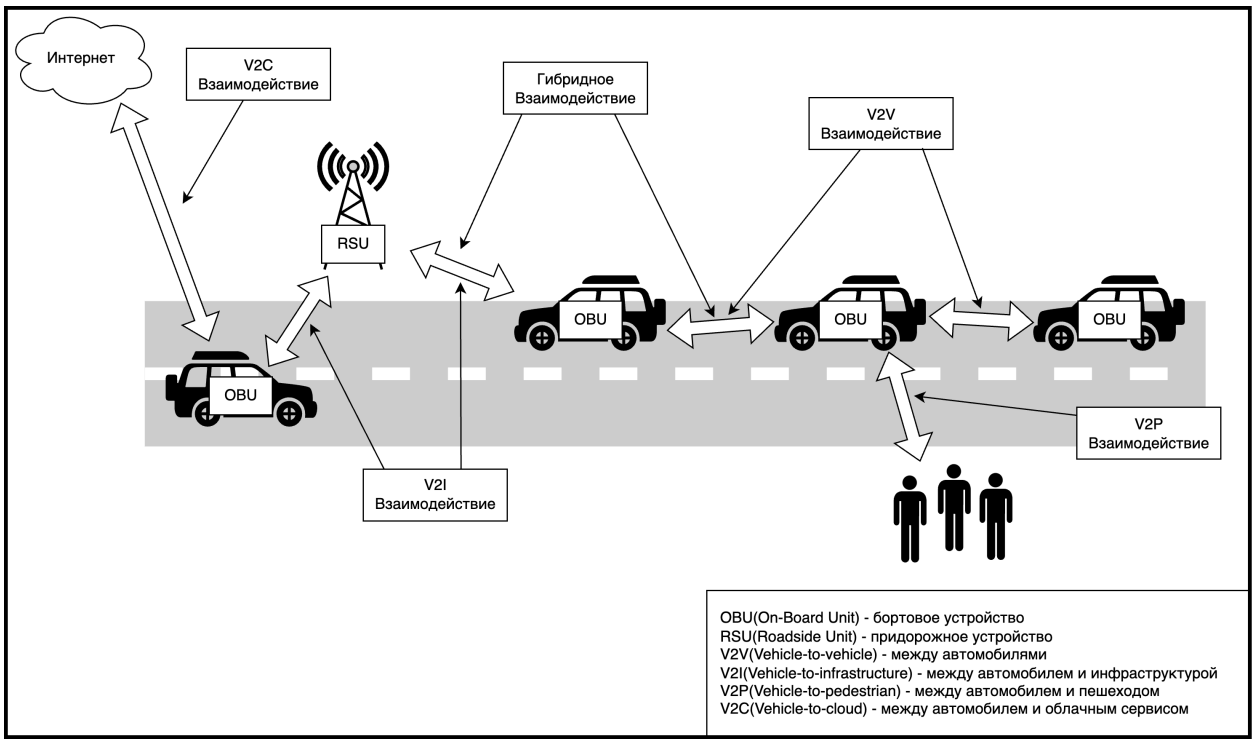

Рис. 1.3: Типы взаимодействий в VANET

оптимизации работы VANET необходимо адаптироваться к динамичной природе сетей и различным требованиям к качеству обслуживания (QoS) для предоставляемых услуг. Важно отметить, что различные приложения в рамках VANET могут иметь разнообразные требования к QoS. К примеру, приложения, не связанные непосредственно с обеспечением безопасности, могут требовать высокой пропускной способности сети, тогда как услуги, направленные на повышение безопасности, должны обеспечивать передачу данных с минимальной задержкой и высокой надёжностью.

В целом, технология VANET способствует созданию более безопасных и эффективных транспортных систем за счёт внедрения развитых средств коммуникации и обмена данными между транспортными средствами и инфраструктурой.

Отличительные особенности VANET

Сети VANET обладают уникальными характеристиками и вызовами, которые делают предоставление услуг и поддержание их надежности сложной задачей. Ниже описаны ключевые аспекты этих характеристик.

Динамичность мобильности

В сетях VANET наблюдается широкий спектр мобильности участников - от стационарных элементов, таких как RSU, до медленно и быстро передвигающихся транспортных средств, что создает трудности для эффективного обмена данными внутри сети.

Ограничения движения

Мобильность узлов сети VANET ограничена инфраструктурой транспортной сети, которая различается в зависимости от географического расположения и характера городской планировки, влияя тем самым на связность и производительность сети.

Частая фрагментация сети

Фрагментация сети VANET обусловлена плотностью и мобильностью узлов. С уменьшением плотности сеть становится более фрагментированной, что негативно сказывается на коммуникации и доставке данных. Быстрая мобильность узлов также вносит динамику в топологию сети, разделяя её на отдельные фрагменты.

Неоднородность

Сеть VANET включает в себя разнообразные узлы с различными функциями и требованиями, варьирующиеся от стационарных элементов до мобильных транспортных средств, которые могут обмениваться как развлекательной информацией, так и критически важными данными для обеспечения безопасности движения.

Масштабируемость

Сети VANET представляются способными охватывать различные географические масштабы, от малых городов до больших городских и масштабов страны, что создает проблемы в обеспечении надежности связи при масштабировании сети. Применение беспилотных летательных аппаратов для улучшения масштабируемости является одним из перспективных направлений развития.

Неограниченные питание и вычислительные ресурсы

В сетях VANET отсутствуют ограничения на мощность и вычислительные возможности благодаря использованию автомобильных аккумуляторов как источника питания и интеграции OBU, что позволяет преодолевать проблемы, связанные с энергопотреблением и обработкой данных.

Проблема дефицита частотного спектра

Стандарты беспроводной связи для транспортных сетей включают технологии Wireless Access in Vehicular Environments (WAVE) и Dedicated Short-Range Communications (DSRC). Однако в ходе исследований в условиях городских сетей с высокой плотностью трафика выявлены вопросы надежности и масштабируемости для сетей на базе DSRC. Основной проблемой является недостаточное количество частотных каналов для обмена данными между автомобилями, что подрывает надежность и возможность расширения сетевых систем. Транспортные средства должны конкурировать за доступ к семи доступным DSRC каналам для обмена информацией, что приводит к перегрузке сети, увеличению времени передачи данных и снижению общей пропускной способности, что отрицательно сказывается на качестве предоставляемых служб.

Влияние окружающей среды

В транспортных сетях VANET коммуникация осуществляется на открытом воздухе, где воздействие окружающей среды на распространение электромагнитных сигналов может быть значительным. Различные объекты, такие как здания, автомобили и деревья, могут создавать помехи для этих сигналов, приводя к многолучевому распространению, замиранию и затуханию сигналов. Также изменение климатических условий может влиять на высокоскоростные соединения VANET, требующие передачи данных с минимальной задержкой.

Точность информационных данных

Использование данных о местоположении от систем навигации, таких как GPS и GNSS, играет ключевую роль в доставке данных в среде VANET. Однако эти системы не всегда могут обеспечить высокую точность определения местоположения, особенно из-за атмосферных явлений в тропосфере и ионосфере. В городских условиях точность GPS может колебаться от 5 до 30 метров в оптимальных условиях, что недостаточно для обеспечения надежности высокоскоростных служб VANET с низкой задержкой. Кроме того, RSU собирают данные о транспортных средствах в их зоне действия, которые далее могут анализироваться облачными сервисами или системами Software-Defined

Networking (SDN). Тем не менее, из-за высокой подвижности и постоянных изменений эти данные могут быть неточными, что снижает эффективность анализа и принятия решений.

Вопросы надежности

Важнейшим аспектом для служб, связанных с безопасностью в сетях VANET [3], является обеспечение связи в режиме реального времени. Любые неточности в передаче данных могут привести к задержкам, что, в свою очередь, способно вызвать существенные проблемы на дорогах, включая серьезные заторы и аварии. Поэтому рекомендуется своевременно принять профилактические меры безопасности, чтобы предотвратить подобные инциденты, давая водителям возможность эффективно реагировать в экстренных ситуациях.

Обеспечение безопасности данных

Для эффективного и надежного управления коммуникациями в сетях VANET необходимо гарантировать безопасность данных. Это включает в себя защиту данных от искажения в процессе передачи, их шифрование таким образом, чтобы предотвратить доступ третьих лиц к информации, а также подтверждение подлинности отправителя сообщений, исключая возможность подделки данных. Безопасность играет ключевую роль, поскольку при нарушении защиты системы транспортные средства могут оказаться под контролем злоумышленников, влекущих за собой дорожные пробки и аварии, потенциально угрожающие жизни. Основными трудностями при интеграции систем VANET являются обеспечение безопасности данных и подтверждение их подлинности, что усложняется из-за динамичной сетевой топологии и мобильности транспортных средств. Ключевое значение имеют методы криптографии, используемые для шифрования и дешифровки информации, требующие эффективного управления криптографическими ключами, что представляет собой сложную задачу из-за частых изменений в сетевой конфигурации VANET. Важным элементом управления ключами является их отзыв, предназначенный для нейтрализации ключей, использованных злоумышленниками, что усложняет процесс из-за роста размеров сети и увеличения количества участников.

Протоколы маршрутизации в сетях VANET

Протоколы маршрутизации в сетях VANET используют технологию многоканальной беспроводной передачи информации от одного узла к другому. Данные протоколы обрабатывают информацию для маршрутизации с целью поддержания соединения, поиска

оптимального пути и его сохранения в таблицах маршрутизации. Разработка протоколов маршрутизации для VANET является задачей для исследований, и на сегодняшний день исследователями и академическими работниками было предложено и проанализировано множество таких протоколов. В зависимости от техники распространения информации, метода обновления пути и пригодности для приложений, протоколы маршрутизации в VANET в основном классифицируются на основе широковещательной передачи (BBR), кластеризации (CBR), топологии (TBR), определения позиции (PBR) и географической (GBR). На рис. 1.4 представлены примеры различных подходов к маршрутизации, используемых для распространения информации в VANET. Протоколы BBR применяют подходы к широковещательной рассылке для реального времени распространения информации, что полезно для приложений интеллектуальных транспортных систем, таких как сообщения о безопасности, дорожных условиях и погоде. BROADCOM, V-TRADE и DECA являются примерами протоколов BBR. Протоколы PBR предполагают, что транспортные средства имеют доступ к услугам GPS для определения своего местоположения и места назначения, используя периодическое маячение для обнаружения соседей в пределах одного прыжка и избежания столкновений. [4]

GPSR, GPCR и A-STAR являются примерами протоколов PBR. Протокол CBR разделяет сети на множество кластеров, каждый из которых состоит из узлов-участников и одного узла-главы кластера. Размер кластера зависит от техник маршрутизации, основанных на местоположении, скорости и направлении движения узла. CBDRP, CBLR и TIBCRPH являются примерами протоколов CBR. Протоколы GBR используют методы многоадресной передачи для доставки данных узлам в определенной географической области, которая называется Зоной Соответствия (ZOR). ROVER, IVG и DRG являются примерами протоколов маршрутизации GBR.

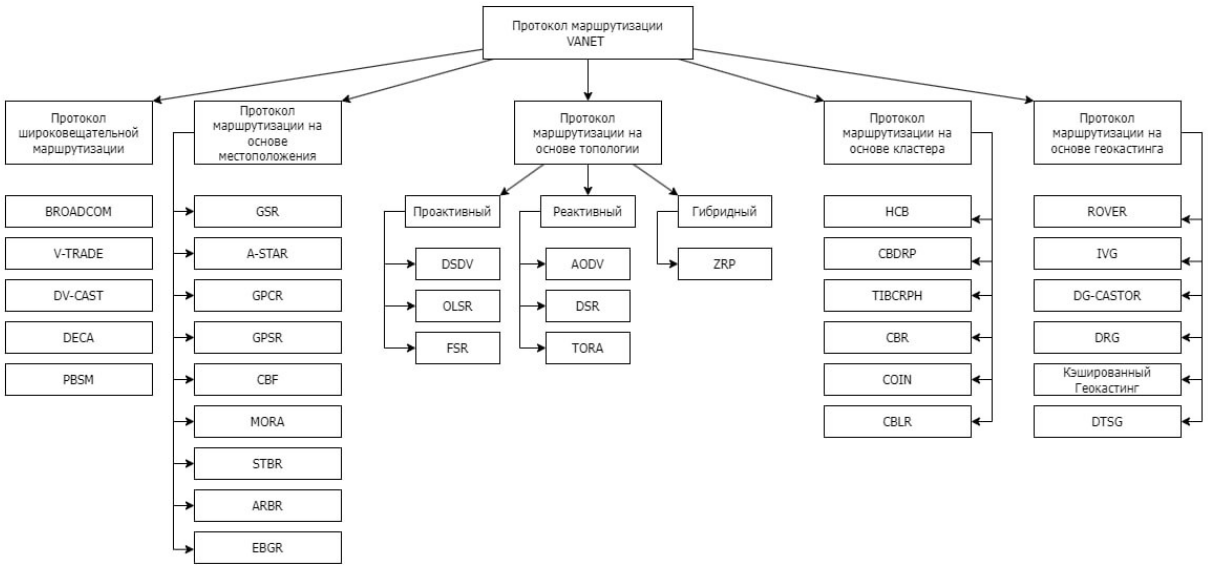

Рис. 1.4: Протоколы маршрутизации VANET

Протоколы маршрутизации, основанные на топологии

Протоколы маршрутизации, основанные на топологии, используют топологическую информацию, установленную в сети, для передачи данных. Эти протоколы можно разделить на три категории: прогнозирующие, реактивные и гибридные протоколы маршрутизации.

Прогнозирующие протоколы маршрутизации регулярно обновляют и поддерживают информацию о маршруте, даже когда маршрут не требуется. Они используют алгоритм кратчайшего пути для поиска маршрута. К примерам прогнозирующих протоколов маршрутизации относятся протоколы маршрутизации по векторам расстояний с последовательностью назначения (DSDV), оптимизированная маршрутизация на основе состояния каналов (OLSR) и маршрутизация с учетом "рыбьего глаза"(FSR).

Реактивные протоколы маршрутизации – это протоколы маршрутизации по требованию, которые поддерживают только недавно использованные пути маршрутизации. Эти протоколы не обновляют все маршруты постоянно. Такой подход позволяет экономить пропускную способность и снижать требования к памяти. В качестве примеров реактивных протоколов маршрутизации можно привести протокол маршрутизации по вектору расстояния "по требованию"(AODV) и динамическую маршрутизацию на основе источника (DSR).

Гибридные протоколы маршрутизации сочетают в себе свойства прогнозирующих и реактивных протоколов маршрутизации. Протокол зонной маршрутизации (ZRP) является примером гибридного протокола маршрутизации.

Проактивные протоколы маршрутизации

В данном типе маршрутизации информация обо всех ассоциированных узлах сохраняется в виде таблиц. Эта таблица регулярно обновляется при каждом изменении топологии за счет обмена управляющими сообщениями. Маршруты определяются на основе этой таблицы. Проактивные протоколы не сталкиваются с начальной задержкой при поиске маршрута, но потребляют значительный объем канала связи для периодического обновления данных о топологии.

Одним из таких протоколов является протокол DSDV.

Протокол DSDV

Протокол маршрутизации DSDV основан на алгоритме Беллмана-Форда и является табличным протоколом маршрутизации. Его разработали Правин Бхагват и Чарльз Э. Перкинс в 1994 году как протокол маршрутизации для мобильных ad-hoc сетей (MANET). Все узлы в сети поддерживают свою таблицу маршрутизации, которая со-

держит запись о назначении, необходимом количестве прыжков для достижения узла назначения и порядковом номере. Таблица маршрутизации DSDV имеет порядковый номер, который используется для предотвращения петель маршрутизации. Этот порядковый номер может быть четным или нечетным в зависимости от доступности связи. Каждый узел использует либо периодические обновления, либо методы обновления по событию для обновления таблицы маршрутизации. Обновления по событию используются каждый раз, когда узел получает пакет DSDV, вызывающий изменения в таблице маршрутизации. Поскольку в DSDV вся информация о маршруте уже доступна, нет необходимости в поиске маршрута, что приводит к меньшей задержке. Однако, поскольку в DSDV вся информация о маршруте содержится в таблице маршрутизации, если топология динамична из-за скорости узлов и увеличения размера сети, DSDV потребляет больше пропускной способности.

Реактивные протоколы маршрутизации

Реактивные протоколы маршрутизации работают по принципу "по требованию то есть маршруты между узлами сети устанавливаются только по запросу. До начала обмена пакетами между источником и пунктом назначения узел инициирует процедуру поиска маршрута, рассылая в сеть управляющий пакет для поиска кратчайшего пути к пункту назначения. Таким образом, данный тип протоколов потребляет меньше пропускной способности по сравнению с проактивными протоколами.

Протокол AODV

Протокол маршрутизации AODV является реактивным и обеспечивает возможность самоорганизации и многопрыжковой связи между мобильными узлами в сетях ad-hoc. Протокол использует порядковые номера назначения для обеспечения маршрутизации без петель. Данный порядковый номер формируется узлом назначения и включается в любую отправляемую информацию о маршруте для указания узлам. Если существуют два пути к узлу назначения, выбирается путь с наибольшим порядковым номером. Благодаря реактивному подходу AODV потребляет мало пропускной способности и памяти. [5]

Поиск маршрута в AODV начинается в порядке запрос-ответ. Исходный узел транслирует сообщение с запросом маршрута (RREQ) всем своим соседям в пределах одного прыжка для его обнаружения. Соседние узлы переотправляют это сообщение RREQ, пока оно не достигнет узла назначения. В ответ узел назначения посылает сообщение с ответом на запрос маршрута (RREP) по обратному маршруту через промежуточные узлы до исходного узла. Сообщение о ошибке маршрута (RERR) транслируется узлом, который потерял свой путь к следующему прыжку.

Протокол DSR

DSR является реактивным протоколом и работает аналогично AODV. Однако отличие заключается в том, что AODV сохраняет в своей таблице маршрутизации только путь до следующего перехода, в то время как DSR сохраняет в своей таблице маршрутизации полный путь от источника к пункту назначения. DSR начинает поиск маршрута только по требованию. Исходный узел определяет полную последовательность узлов, через которые должен пройти пакет, и вставляет эту последовательность в каждый отправляемый пакет данных. Эта последовательность используется каждым узлом в маршруте для определения следующего реле до пункта назначения.

Глава 2

Моделирование сети VANET

2.1 Описание модели

Суть эксперимента

В данной работе будет проведён эксперимент, в рамках которого будет производится моделирование сети VANET.

Схема эксперимента показана на рис. 2.1.

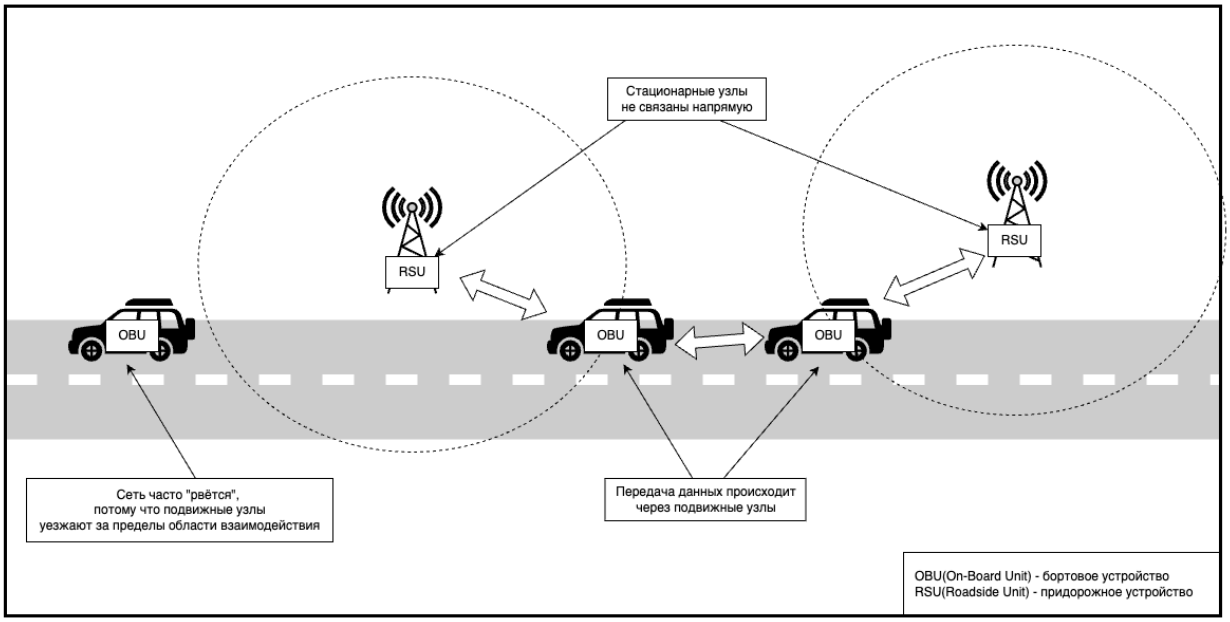

Рис. 2.1: Схема эксперимента

На какой-то местности расположены несколько стационарных узлов с беспроводным передатчиком связи (RSU). Мощность датчиков и расстояние между узлами таковы, что стационарные узлы не могут достать друг до друга и передать какие-либо данные непосредственно.

Моделируемая местность имеет автомобильную дорогу, по которой передвигаются подвижные узлы с бортовым беспроводным передатчиком связи (OBU). Подвижные

узлы передвигаются по правилам автомобильного дорожного движения на моделируемом участке дороги.

В какой-то момент симуляции, когда подвижные узлы изменят топологию сети таким образом, что между стационарными узлами образуется путь(включающий подвижные узлы), начнётся передача данных из одного стационарного узла в другой. При этом путь, по которому следуют данные, будет неизбежно меняться с течением времени симуляции, т.к. подвижные узлы почти непрерывно передвигаются и меняют топологию сети. Подвижные узлы могут так же пропадать из зоны доступности стационарных узлов.

В такой нестабильной сети, когда топология сети постоянно "рвётся" предлагается проверить эффективность трёх протоколов маршрутизации: AODV, DSDV, DSR. Принципы моделирования сети и анализа эффективности протоколов будут рассмотрены в этой главе.

Развертывание и тестирование сетей VANET связаны с высокими затратами и требуют значительных усилий. В качестве альтернативного решения, имитационное моделирование является полезной и менее затратной альтернативой перед фактической реализацией. Для достижения хороших результатов при моделировании VANET необходимо разработать точные модели, что является непростой задачей ввиду сложностей инфраструктуры VANET (например, симуляторам необходимо моделировать как образцы мобильности, так и протоколы коммуникации) [6, 7, 8].

Подробно процесс моделирования показан в статье [9]. Автор использует набор инструментов для моделирования NS-2, SUMO и OSM и это окружение кажется достаточно удобным и быстрым в освоении и настройке.

Моделирование V2V взаимодействия между узлами сети VANET подробно проанализарован в [10]. Сравниваются три протокола маршрутизации AODV, DSR и DSDV. Так же автор объясняет разницу между этими протоколами и особенности их работы в условиях подвижных узлов сети, свойственных для VANET.

Можно сформировать общую схему имитационной модели сети VANET, которую разделим на две части: моделирование мобильности и моделирование сети.

Первая часть - моделирование мобильности, включает в себя определение количества узлов сети и описание их местоположения в каждый момент времени. Вторая часть - моделирование сети, содержит определение конфигурации сети и описывание сетевых взаимодействий узлов в каждый момент времени симуляции.

При наложении смоделированной сети на мобильность узлов получается модель сети VANET.

Общая схема моделирования сети изображена на рис. 2.2 и будет рассмотрена подробнее далее.

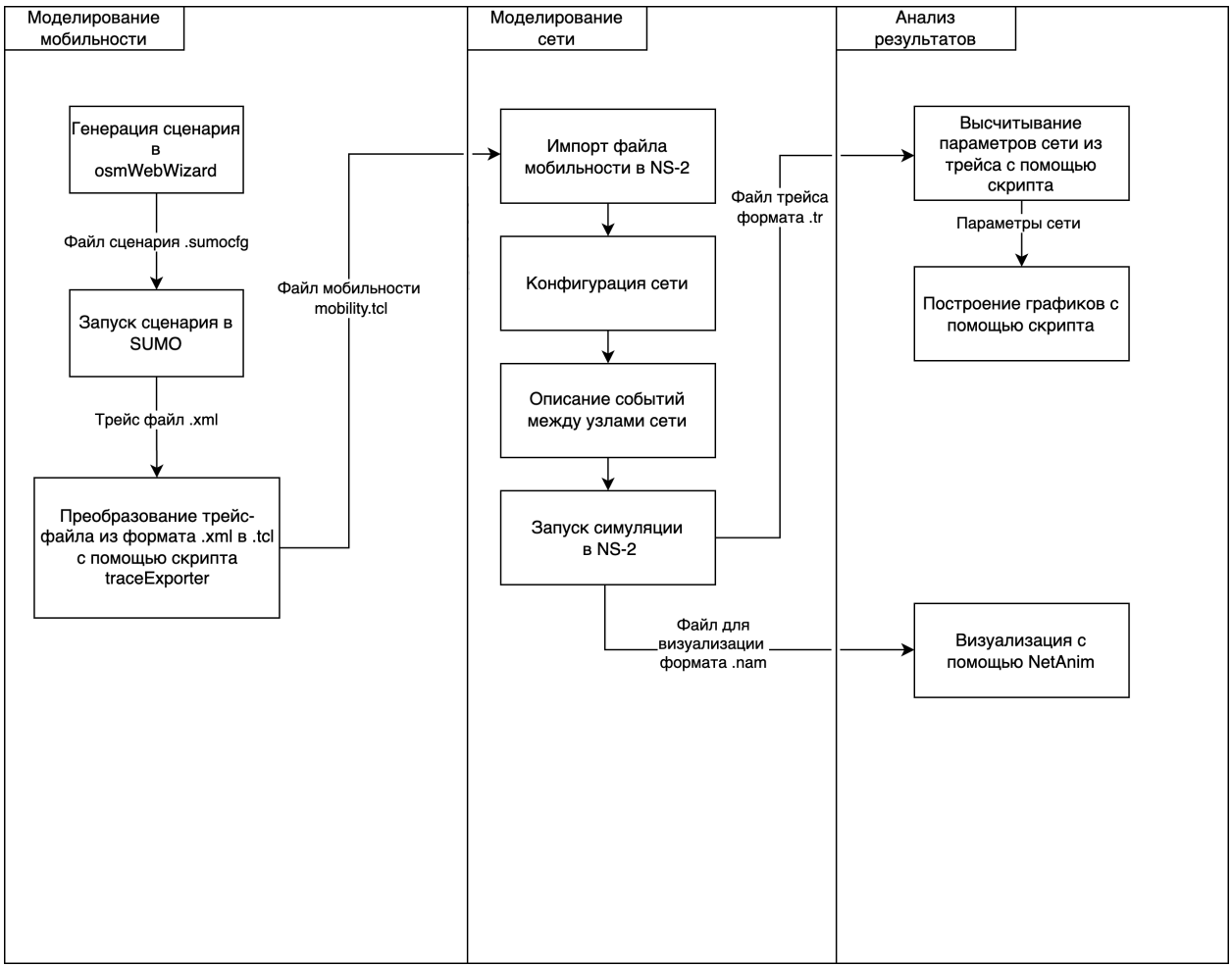

Рис. 2.2: Схема моделирования

Моделирование мобильности в сетях VANET

Критически важным аспектом в исследовании симуляции сетей VANET является необходимость в модели мобильности, которая соответствовала бы реальному поведению транспортных средств в дорожном движении. Симуляторы мобильности в основном используются для генерирования закономерностей движения транспортных средств по определённому маршруту. В контексте моделирования мобильности транспортных средств различают макромобильность и микромобильность. При моделировании макромобильности симуляторам необходимо учитывать все макроскопические аспекты, влияющие на дорожное движение: топологию дорог, ограничения движения автомобилей, ограничения скорости, количество полос, правила безопасности и знаки дорожного движения, регулирующие правила проезда на перекрестках. Микромобильность, с другой стороны, относится к индивидуальному поведению водителей при взаимодействии с другими участниками дорожного движения или с дорожной инфраструктурой: скорость движения в различных дорожных условиях, ускорение, замедление и критерии обгона, поведение на перекрёстках и при наличии дорожных знаков, общее отношение к вождению, связанное с возрастом, полом или настроением водителя. Идеальная

симуляция в сетях VANET должна учитывать как описания макро-, так и микромобильности. Примеры симуляторов мобильности включают SUMO, VISSIM, SimMobility, PARAMICS и CORSIM.

В данной работе будет подробнее рассмотрен симулятор мобильности SUMO.

Симулятор городской мобильности SUMO

SUMO — это пакет программ для моделирования трафика с открытым исходным кодом. SUMO используется для исследования различных аспектов движения - от выбора маршрута и алгоритмов работы светофоров до моделирования взаимодействия транспортных средств. Этот инструментарий применяется в различных проектах для моделирования автоматизированного вождения или стратегий управления трафиком, предлагая функции, такие как движение транспортных средств в непрерывном пространстве и дискретном времени, множественные типы транспортных средств, много-полосное движение с возможностью смены полосы, различные правила приоритета проезда, светофоры, удобный графический интерфейс, быстрое выполнение и взаимодействие с другими приложениями в реальном времени.

Генерация сценария

Есть много способов создать файл симуляции, но один из самых удобных – это использовать OsmWebWizard.

Схема, по которой может осуществляться моделирование мобильности в сетях изображена на рис. 2.3.

OsmWebWizard это программа, позволяющая создать файл симуляции SUMO через графический интерфейс. При этом программа предоставляет возможность просимулировать практически любое место на карте мира благодаря использованию OpenStreetMap в процессе генерации сценария.

Сама программа OsmWebWizard поставляется вместе с кодом SUMO и доступна после установки SUMO. Она представляет из себя скрипт на языке Python и легко запускается через кроссплатформенный интерпретатор Python.

Для симуляции сети VANET зачастую не требуется моделировать никакие дороги кроме автомобильных и элементы городской инфраструктуры, поэтому их можно убрать с помощью графического интерфейса OsmWebWizard.

Так же можно настраивать количество автомобилей, участвующих в симуляции и плотность трафика.

Нажатием на кнопку "Generate scenario"создаётся файл с настройками симуляции для SUMO в формате .sumocfg

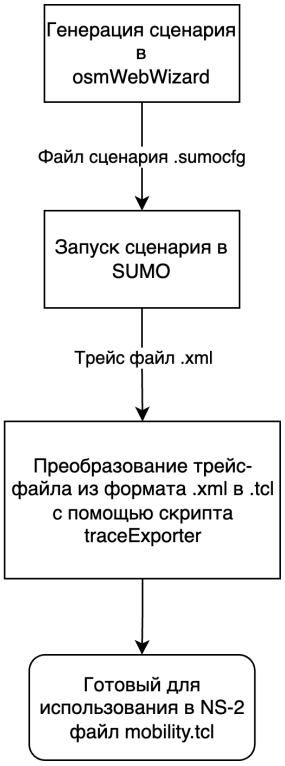

Рис. 2.3: Схема моделирования мобильности

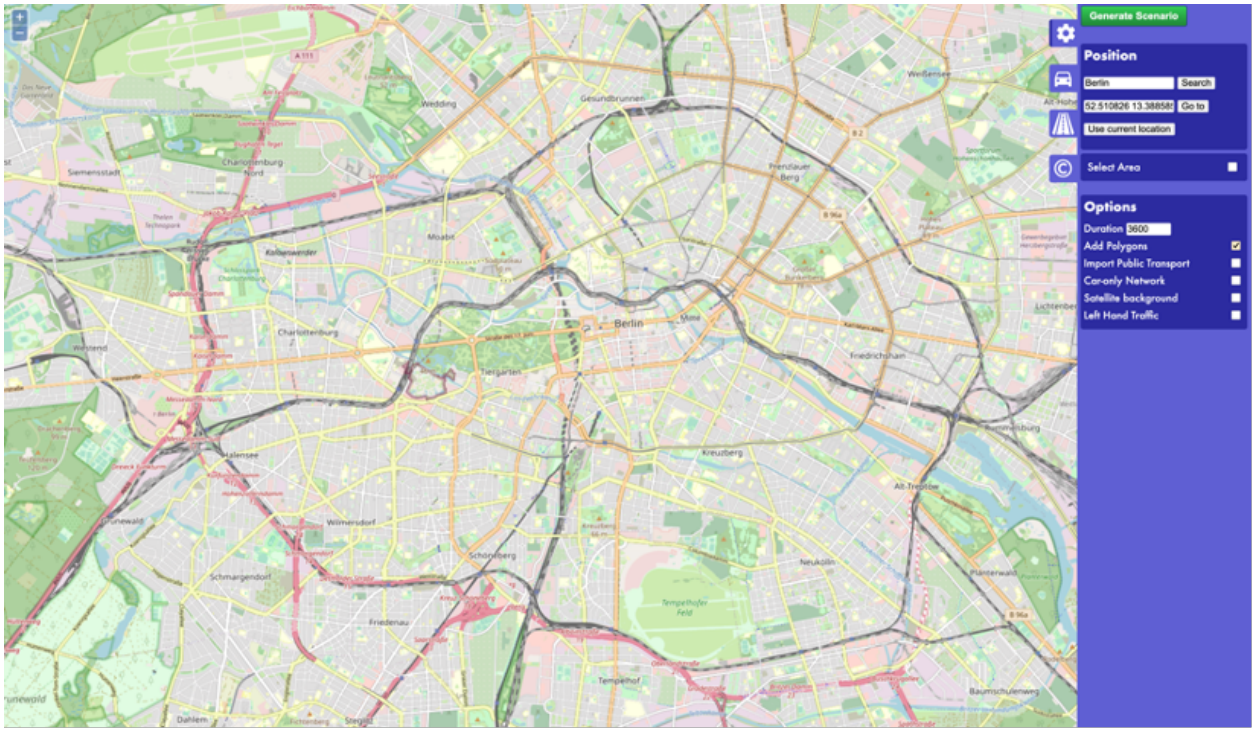

Рис. 2.4: Интерфейс программы OsmWebWizard

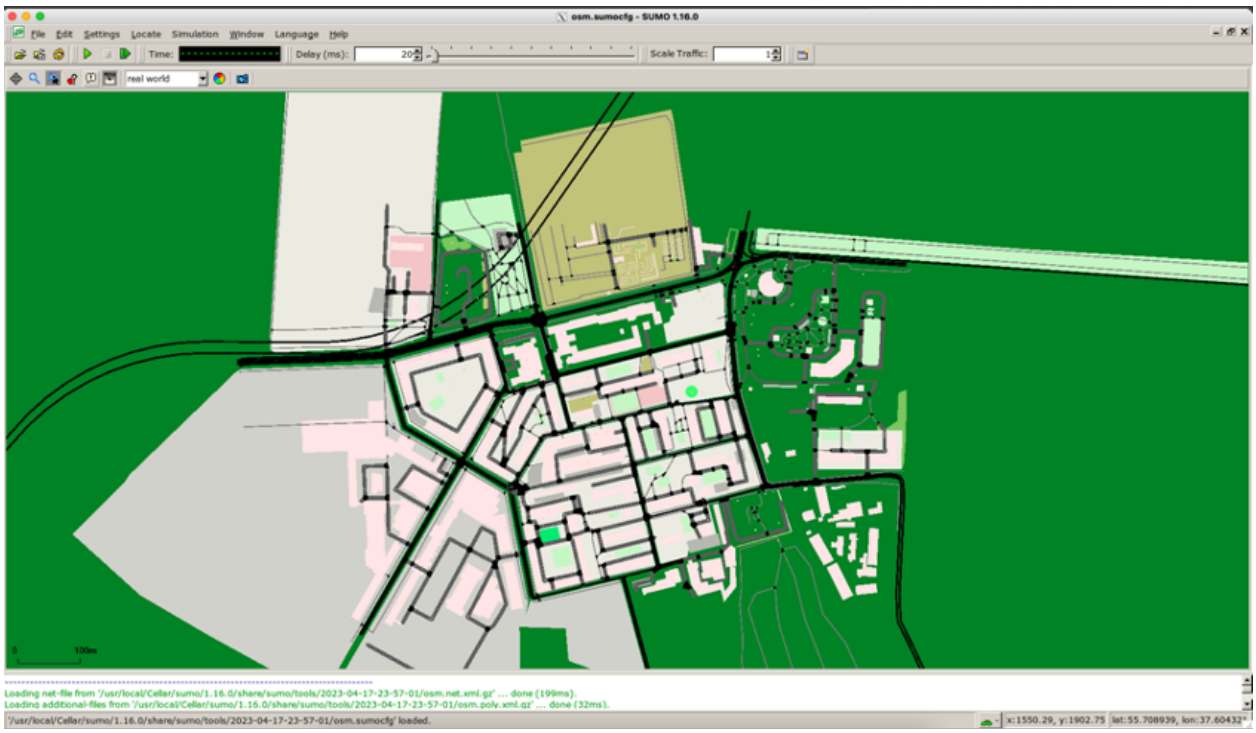

Рис. 2.5: Интерфейс программы SUMO

Запуск симуляции в SUMO

SUMO можно использовать как через графический интерфейс, так и через интерфейс командной строки. В зависимости от необходимого результата, нужно использовать тот или иной вариант. Если требуется визуально понаблюдать за симуляцией, то запускается графический интерфейс, а если есть необходимость только получить результаты симуляции, то удобнее использовать командный интерфейс.

Запуск симуляции может осуществляться с помощью импортирования файла .sumocfg, например, полученного из OsmWebWizard, что является удобным способом быстрого запуска симуляции мобильности.

Результатом запуска симуляции в SUMO является файл с описанием мобильности всех участвующих в симуляции узлов.

Преобразование файла мобильности

Чтобы использовать результат симуляции SUMO, необходимо преобразовать его в формат, подходящий для NS-2(эта программа будет рассмотрена далее). Для этого можно использовать скрипт написанный на языке Python traceExporter.py, он поставляется так же вместе с SUMO.

Принцип работы этого скрипта прост. Он преобразует все записи о перемещении автомобилей в команды на языке NS-2 otcl, предварительно превращая автомобили в узлы сети. Таким образом, получаемый на выходе файл, по сути является набором

команд на языке otcl, отвечающим за создание и перемещение конкретных узлов сети во времени.

После запуска скрипта получается файл mobility.tcl, который можно легко импортировать в NS-2. При импорте все команды, содержащиеся в файле mobility.tcl, выполнятся и события перемещения узлов, описанные в файле, будут зарегистрированы. При старте симуляции в каждый момент вермени будут выполняться все команды, зарегистрированные на этот момент времен и узлы будут перемещаться соответствующим образом, как это происходило в симуляции SUMO.

Моделирование сети

Симулятор сети применяется для моделирования обмена сообщениями между соединенными узлами. В контексте VANET это обычно включает в себя бортовые устройства транспорта (OBU) и стационарные придорожные устройства (RSU) и в большинстве случаев связано с беспроводной коммуникацией. В идеале моделируются все компоненты коммуникационной системы (например, весь стек протоколов), и в конечном итоге симуляция также включает другие соответствующие метрики (например, отношение сигнал/шум, показатели ошибок пакетов). Модель сети описывает как компоненты сети, так и события. Узлы, маршрутизаторы, коммутаторы и связи являются примерами компонентов. События, в свою очередь, могут включать передачи данных и ошибки пакетов.

Для данного сценария симуляции выходные данные от симулятора сети обычно включают метрики, связанные с сетью, соединениями и устройствами. Также обычно доступны файлы трассировки. Такие файлы записывают каждое произошедшее событие в симуляции и могут быть обработаны для дальнейшего анализа. Большинство доступных симуляторов сетей базируются на дискретно-событийном моделировании. В этом подходе хранится список "ожидающих событий", которые затем обрабатываются по порядку на каждом шаге симуляции. Некоторые события могут инициировать новые. Например, прибытие пакета на узел может спровоцировать отправку нового пакета. Примеры доступных симуляторов сетей (некоторые из них широко используются в сетях VANET) включают OMNeT++, OPNET, JiST/SWANS, NS-3 и NS-2.

В данной работе подробнее будет рассмотрен сетевой симулятор NS-2.

Сетевой симулятор NS-2

Network Simulator 2 (NS-2) представляет собой программный комплекс на основе объектно-ориентированной парадигмы, разработанный для моделирования различных аспектов сетевых систем и процессов. Сердцевиной системы является ядро, написанное на языке программирования C++, а для сценариев и интерфейса пользователя применяется

язык Object oriented Tool Command Language (OTcl). Наличие классовой структуры и иерархии, которая поддерживается обоими языками, позволяет удобно организовать моделирование, сохраняя при этом однозначное соответствие между элементами разработки на разных уровнях.

Платформа NS-2 обеспечивает поддержку широкого круга сетевых протоколов, что делает её весьма полезным инструментом в изучении и анализе современной сетевой инфраструктуры. Включает в себя реализации протоколов на всех уровнях сетевого обмена данных, таких как MPLS, IPv6, OSPF, и RSVP, а также несколько механизмов управления очередями (например, RED, WFQ, CBQ, SFQ), что позволяет имитировать работу реальных сетевых условий. Особое внимание уделено поддержке протоколов для беспроводных сетей, в том числе AODV, DSDV, DSR, расширяя область применения инструмента.

Среди ключевых особенностей NS-2 выделяется его гибкость в имитации различных типов трафика, включая трафик с Пуассоновским распределением и самоподобный трафик. Эта возможность дополнена способностью пользователя создавать собственные математические модели и функции, используя C++. К тому же, NS-2 позволяет моделировать ошибки на канальном уровне в процессе передачи данных, такие как искажение и потери пакетов, предлагая разнообразные методы для задания характеристик ошибок.

Для анализа и интерпретации результатов моделирования в NS-2 интегрированы инструменты визуализации, включая Network Animator (NAM) и Xgraph. NAM позволяет наглядно представить динамику сетевого взаимодействия, включая топологию, потоки данных и работы сетевого оборудования, на основе данных трассировки. Xgraph же облегчает анализ статистики сетевой активности, позволяя строить графики непосредственно изскриптов моделирования.

Программный комплекс NS-2, разрабатываемый как программное обеспечение с открытым исходным кодом, доступен для бесплатного использования, модификации и распространения. Это обеспечивает его высокую приспособляемость к специфическим исследовательским задачам и различным сетевым сценариям. Поддержка множества операционных систем, включая Linux, OS X, Solaris, FreeBSD и Windows, увеличивает удобство использования NS-2 в различных средах разработки и обеспечивает широкую доступность для пользователей, независимо от их предпочтений в операционных системах.

Важным аспектом, способствующим популярности и функциональному развитию NS-2, является активное сообщество разработчиков и пользователей. Это сообщество обеспечивает постоянное обновление и расширение возможностей программного комплекса, разработку новых модулей и протоколов, а также предоставление документа-

ции, руководств и разнообразных учебных материалов. Такой открытый подход способствует повышению качества и эффективности как самого программного обеспечения, так и проводимых с его помощью исследований.

Принцип моделирования сети в симуляторе NS-2

В создании сценария симуляции в NS-2 можно выделить несколько этапов и частей.

1. Конфигурация параметров сети и узлов
2. Перемещение узлов сети
3. Генерация трафика в сети
4. Отслеживание событий и запись их в трейс-файлы
5. Завершение симуляции

Конкретный пример написания программы на языке Otcl будет рассмотрен в Главе 3, а сейчас предлагается рассмотреть суть каждого этапа в теории.

Конфигурация параметров сети и узлов

Этап конфигурирования сети необходим, чтобы определить все правила, по которым будет протекать симуляция. Этот этап включает в себя настройку следующих параметров:

1. Simulation area - Область моделирования. Определяет размеры географической области, в которой будет происходить симуляция VANET. Важен для определения условий, в которых будут функционировать транспортные средства, и для реалистичности моделирования движения наземных транспортных средств. Пример возможного значения: 1000m x 1000m - означает, что область моделирования представляет собой квадрат со стороной 1000 метров.
2. MAC Type - Тип MAC (Media Access Control - Управление доступом к среде). Описывает метод управления доступом к среде передачи данных, что критически важно для эффективной и справедливой передачи данных между узлами в сетевых условиях, характерных для VANET. Пример возможного значения: 802.11, это выбор протокола беспроводного доступа, используемого в сетях Wi-Fi, который является общепринятым для VANET.

3. N/W Interface Type - Тип сетевого интерфейса. Определяет особенности и характеристики сетевого интерфейса, используемого узлами для обмена данными. Имеет значение для точности моделирования взаимодействия между устройствами сети. Пример возможного значения: Phy/WirelessPhy, выбор указывает на использование физического уровня беспроводной сети.

4. Interface Queue Type - Тип очереди интерфейса. Указывает на способ управления очередями пакетов в сетевом интерфейсе, что, в свою очередь, влияет на производительность сети и качество обслуживания. Пример возможного значения: Queue/DropTail. Этот тип означает использование очереди с "отбрасыванием по принципу последний пришел - первый ушел которая подразумевает, что новые пакеты будут отбрасываться, если в очереди нет свободного места.

5. Propagation model - Модель распространения. Имитирует способ распространения радиосигналов в среде, что несет критическую важность для обеспечения реалистичности моделирования коммуникационной среды в VANET. Пример возможного значения: TwoRayGround. Модель распространения "Двухлучевая наземная"учитывает как прямую линию видимости, так и отражение от земли.

6. Transmission range - Дальность передачи. Определяет максимальное расстояние, на котором принимающий узел может успешно обнаруживать передаваемые сигналы. Этот параметр существенно влияет на структуру и свойства сети. Пример возможного значения: 250m.

7. Antenna model - Модель антенны. Описывает характеристики антенны, используемой для передачи и приема радиосигналов, что определяет эффективность связи внутри сетевого пространства. Пример возможного значения: OmniAntenna - омнидирекциональная антенна, обеспечивающая равномерное распространение сигнала во всех направлениях.

8. Number of nodes - Количество узлов (транспортных средств). Определяет число участвующих в сети VANET транспортных средств, что критически важно для изучения масштабируемости и производительности сети.

9. Routing protocols - Протоколы маршрутизации. Определяют алгоритмы построения маршрутов между узлами, что влияет на эффективность и надежность коммуникации в сети. Пример возможного значения: AODV - протокол маршрутизации, рассмотренный выше в Главе 1.

10. Transport protocols - Транспортные протоколы. Указывают на протоколы уровня транспорта, используемые для контроля передачи данных; важны для обеспечения

достоверности, управления потоком и устранения ошибок в процессе передачи данных между узлами. Пример возможного значения: TCP или UDP. Выбор зависит от необходимости контроля над ошибками и потоком данных; TCP обеспечивает доставку без ошибок, в то время как UDP обеспечивает более быструю передачу без подтверждения получения.

11. Traffic Type - Тип трафика. Классифицирует виды трафика (например, голосовой, видео, передача файлов), что позволяет более точно моделировать поведение сети под различными видами нагрузки. Пример возможного значения: FTP, означает, что будет использоваться передача файлов.

12. Packet size - Размер пакета. Определяет размеры передаваемых данных в байтах за одну операцию отправки, что непосредственно влияет на производительность сети и эффективность использования канала передачи данных.

13. Transmission rate - Скорость передачи. Устанавливает максимально возможную скорость передачи данных между узлами сети в битах в секунду, что является ключевым параметром для анализа пропускной способности сети.

14. Simulation time - Время симуляции. Определяет продолжительность симуляционного эксперимента в секундах, что важно для оценки динамических характеристик и стабильности сетевых процессов на протяжении времени.

Некоторые из этих параметров необходимо явно задавать, а какие-то из параметров подходят для симуляции в своём значении по умолчанию. Бывают параметры, которые сложно задать явно и это зачастую делается косвенно за счёт изменения логики работы элементов сети. Например - параметр дальности передачи может контролироваться мощностью передатчика узла, которую можно явно задать. Уменьшив мощность передатчиков - понизится и расстояние, на котором узлы смогут взаимодействовать друг с другом.

Перемещение узлов сети

В работе рассматривается не произвольная симуляция сети в NS-2, а конкретно симуляция мобильных самоорганизующихся сетей - VANET. Соответственно, важным этапом симуляции является перемещение узлов. Как уже было описано ранее, по сути своей любое перемещение узлов - это какое-то событие в NS-2. Принцип достаточно прост - создаётся узел специальной командой, узлу задаются начальные координаты.

Для перемещения используется специальная команда setdest, которая выполняет перемещение узла в конкретную точку в конкретный момент времени.

Таким образом необходимо сформировать список команд, перемещающих узлы в определённые моменты времени в определённые точки, задавая этим скорость узлов и их мобильность. В общем и целом это делается на этапе моделирования мобильности, как уже было рассмотрено.

Генерация трафика в сети

Этот этап может быть устроен по разному в зависимости от требуемых результатов. В случае рассматриваемого в работе эксперимента - предлагается генерировать трафик из одного стационарного узла в другой.

В NS-2 для этих целей используются агенты. В контексте моделирования сетей в симуляторе NS-2, агенты являются критически важными компонентами, представляющими конечные точки коммуникации или источники трафика внутри сетевой модели. Они инкапсулируют различные протоколы соединения, предоставляя механизмы для инициации, управления и завершения коммуникационных сессий. Агенты в NS-2 делятся на различные категории в зависимости от их функций и протоколов, к которым они относятся. Конкретно, Agent/TCP/Newreno и Agent/TCPSink представляют функционал для установки TCP соединения между узлами сети.

Agent/TCP/Newreno в NS-2 представляет собой реализацию протокола передачи данных TCP с использованием алгоритма управления перегрузкой сети по методу NewReno. NewReno является модификацией алгоритма Reno, предоставляя улучшенное восстановление после потери пакетов, что особенно важно в динамичных сетевых условиях, характерных для VANET. Агент выполняет задачи по надежной передаче данных, корректному управлению размером окна перегрузки и быстрому восстановлению после обнаружения потери пакетов.

Агент Agent/TCPSink, в свою очередь, предназначен для работы в паре с агентом TCP в качестве получателя данных. Он действует как конечная точка, которая принимает данные, отправляемые TCP-агентом, и обрабатывает подтверждения (ACKs) для отправленных пакетов. Это позволяет симулировать полноценный цикл TCP-соединения, включая механизмы управления перегрузкой и управления потоком. TCPSink играет важную роль в обеспечении точности моделирования поведения сетевого протокола, позволяя изучать различные аспекты работы и эффективности TCP в условиях, близких к реальности.

Чтобы установить TCP соединение между двумя узлами необходимо командой attach-agent прикрепить к двум узлам агенты Agent/TCP/Newreno и Agent/TCPSink для отправителя и получателя соответственно. Затем вызывается команда connect между двумя этими агентами.

Но генерации трафика не происходит. Для этого необходимо поверх TCP отправлять

какой-то набор данных. Для этих целей используются другого рода агенты - Application агенты.

Примеры таких агентов:

1. Application/Telnet - Имитирует трафик от приложения Telnet, которое часто используется для удаленного доступа к серверам. Трафик Telnet характеризуется нерегулярными порывами данных, в зависимости от активности пользователя.

2. Application/Ping - Эмулирует генерацию трафика эхо-запросов ICMP (Internet Control Message Protocol), как при использовании команды ping для проверки доступности хоста на сетевом соединении. Может использоваться для измерения задержек в сети и потери пакетов.

3. Application/FTP - служит для моделирования поведения протокола передачи файлов (File Transfer Protocol - FTP) в рамках сетевого трафика. Протокол FTP используется для передачи файлов между клиентом и сервером в сети Интернет или локальной сети. В контексте симуляции сетей NS-2, агент Application/FTP позволяет исследователям точно эмулировать процесс передачи файлов и оценивать влияние различных сетевых параметров и стратегий на эффективность и надёжность этого процесса.

Предлагается использвоать в эксперименте агент Application/FTP. Чтобы начать генерацию трафика, необходимо прикрепить агент Application/FTP к инстансу агента Agent/TCP/Newreno, который является источником данных. Чтобы начать генерацию трафика поверх TCP, необходимо использовать команду start.

Отслеживание событий и запись их в трейс-файлы

Этот этап легко реализуется в NS-2 встроенными средствами. Существует команда, начинающая запись всех событий передачи данных в трейс-файлы - trace-all. Кроме того, для создания анимации так же требуется запись в определённом формате данных о симуляции в отдельный трейс-файл. Для этих целей используются две команды: namtrace-all и namtrace-all-wireless.

Вызов этих трёх команд даст на выходе из симуляции два трейс-файла.

Завершение симуляции

Этап завершения симуляции представляет собой технические действия по завершению всех процессов в симуляции и записи всех необходимых трейс-файлов. В целом теоретических знаний тут не требуется, только конкретные команды, которые будут рассмотрены в Главе 3.

2.2 Методология анализа результатов

Как видно на рис. 2.3, после запуска симуляции NS-2 имеются два артефакта - трейс-файл движения пакетов и файл анимации.

Каждый из них можно проанализировать соответствующим способом.

Визуальный анализ

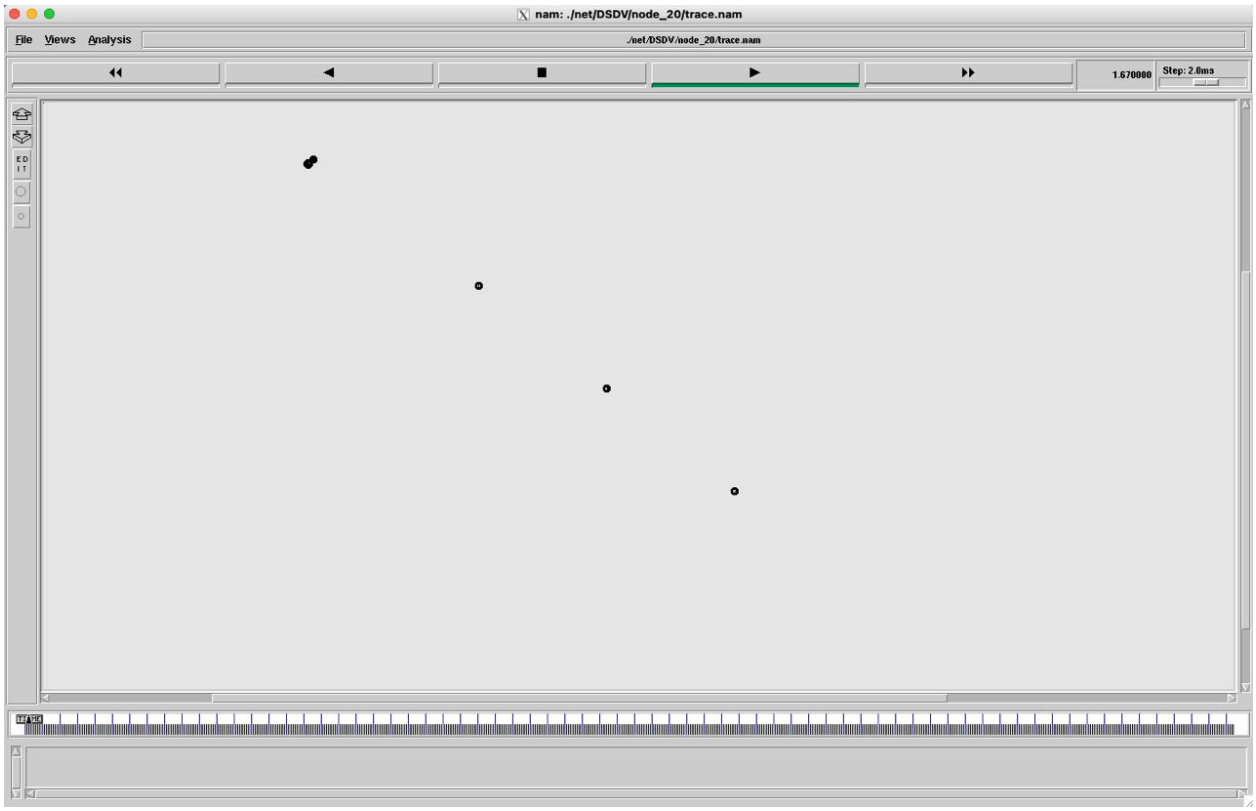

Рис. 2.6: Интерфейс NetAnim

Первый и самый очевидный способ анализа - визуальный. Вместе с NS-2 в архиве "all-in-one"поставляется программа NetAnim. Это програма для визуального отображения процесса симуляции.

В скрипте NS-2, как уже было рассмотрено ранее, присутствует специальная команда namtrace-all, которая записывает все события в файл анимации.

После запуска симуляции, с помощью NetAnim можно просмотреть анимацию из полученного файла в формате .nam.

Интерфейс NetAnim изображён на рис. 2.6.

При запуске анимации - подвижные узлы сети начинают передвигаться и посылать пакеты. Узлы в интерфейсе изображены в виде круга, содержащего внутри себя номер узла, который был задан в скрипте NS-2.

Так как используется беспроводной канал, широковещательные сообщения изображены расплывающимися окружностями от узла.

Направленные пакеты к определённому узлу изображены отрезком, перемещающимся в пространстве от источника к приёмнику.

С помощью визуального наблюдения можно быстро определить те проблемы, которые труднее обнаружить с помощью анализа трейс-файла.

Например, наблюдая за анимацией, можно обнаружить, что прямой связи между стационарными узлами не существует и пакеты начинают передаваться только когда между стационарными узлами появляется подвижный узел.

Пример кадра из анимации можно увидеть на рис. 2.7.

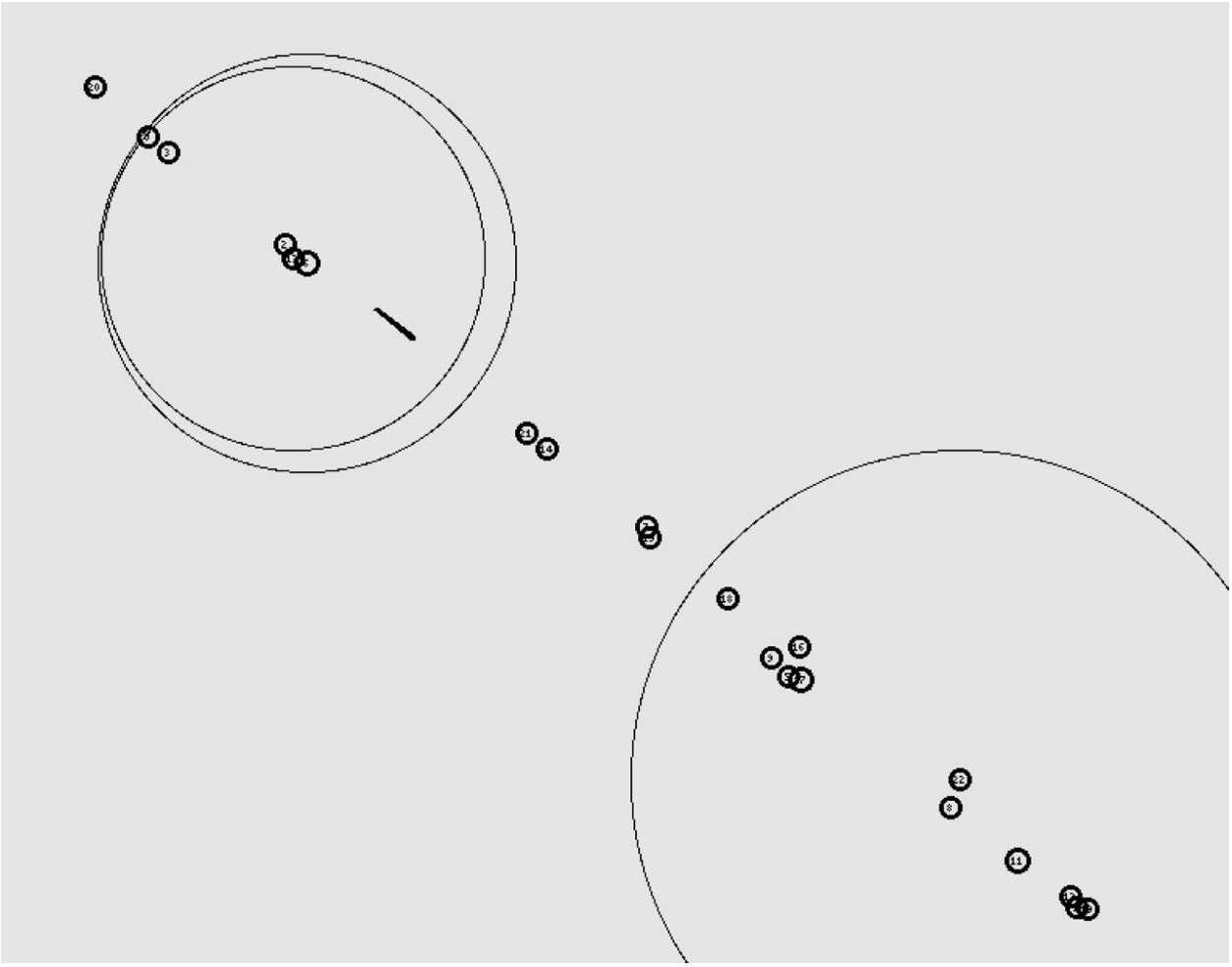

Рис. 2.7: Кадр из анимации NetAnim

Структура трейс-файла

Чтобы проводить не визуальный анализ сети, необходимо разобраться в том, как устроен второй полученный артефакт - трейс-файл.

Для примера будет рассмотрена 16 строка из трейс-файла, часть содержимого которого можно видеть на рис. 2.8.

Это позволит разобраться в том, как устроен трейс-файл и придумать принцип его анализа.

```
M 9.00000 0 (148.90, 550.79, 0.00), (166.42, 551.39), 17.57
M 10.00000 0 (166.42, 551.39, 0.00), (183.17, 551.97), 16.79
s 10.000000000 _0_ AGT  --- 0 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 10.000000000 _0_ RTR  --- 0 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 10.000000000 _0_ RTR  --- 0 AODV 48 [0 0 0 0] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 10.000948256 _15_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 10.000948296 _38_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 10.000948510 _43_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 10.000948510 _26_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 10.000948510 _21_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 10.000948510 _19_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 10.000948510 _6_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 10.000948547 _12_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 10.000948553 _32_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 10.000948553 _36_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
s 10.001376455 _6_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [6:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
s 10.002086636 _32_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [32:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 10.002484455 _19_ RTR  --- 0 AODV 48 [0 ffffffff 6 800] ------- [6:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
```

Рис. 2.8: Содержимое трейса ns2

r 10.000948296 _38_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST). Эта строка из трейс-файла NS-2 содержит информацию об одном пакете, который был отправлен в сеть. Расшифровка элементов строки:

1. r - тип события, в данном случае это отправка пакета.
2. 10.000948296 - время отправки пакета в секундах.
3. _38_ - идентификатор узла, который отправил пакет.
4. RTR - тип узла, который отправил пакет (в данном случае это маршрутизатор).
5. — — — - идентификатор узла, который должен получить пакет (в данном случае это широковещательный адрес).
6. 0 - идентификатор канала, по которому был отправлен пакет.
7. $AODV$ - протокол маршрутизации, который использовался для отправки пакета.
8. 48 - размер пакета в байтах.
9. [0$ffffffff$0800] - информация о маршруте, который будет использоваться для доставки пакета. В данном случае это начальный пакет, поэтому маршрут не известен и все значения установлены в ноль.
10. — — — — — — — — - информация о том, какими типами нод был пройден пакет. В данном случае это еще неизвестно, поэтому все значения установлены в ноль.

11. $[0 : 255 - 1 : 255300]$ - информация о маршруте, который будет использоваться для доставки пакета. В данном случае это начальный пакет, поэтому маршрут не известен и все значения установлены в ноль.

12. $[0x211[10][04]]$ - информация о пакете. $[0x2]$ - это тип пакета, [1] - это идентификатор пакета, [1] - это количество попыток отправки, [10] - это номер сегмента, который был отправлен, [04] - это номер сегмента, который будет отправлен.

13. $(REQUEST)$ - тип пакета, в данном случае это запрос.

Из трейс-файла можно понять состояние каждого пакета в любой момент времени симуляции: размер пакета, пункт назначения, маршрут и т.п.

Это знание позволяет провести анализ производительности сети, обработав и достав нужную информацию о пакетах из трейс-файла.

Подсчёт параметров сети

Чтобы посчитать параметры сети зачастую используются скрипты в формате awk.

Принцип написания этих файлов прост, каждая строка трейс-файла разделяется на части. Как было указано выше, данные в строке трейс-файла разделены пробелами, поэтому после разделения строки пробелами получается массив данных. Каждый элемент в зависимости от типа сообщения содержит разные данные.

Предлагается рассчитать несколько интересных для исследования параметров сети:

1. PDR(Packet Delivery Ratio) - процент доставки пакетов.

2. Delay - время, требующееся для доставки пакета внутри сети.

3. Throughput - пропускная способность сети.

4. NRL(Normalized Routing Load) - метрика, показывающая сколько пакетов из всех отправленных служит целям роутинга, а не передаче данных.

5. Packets sent - количество отправленных пакетов внутри сети.

Packet Delivery Ratio

Коэффициент успешной доставки пакетов (Packet Delivery Ratio, PDR) определяется как отношение числа пакетов, которые были успешно доставлены к месту назначения, к числу пакетов данных, отправленных из источника. Этот показатель вычисляется путем деления количества пакетов, полученных пунктом назначения, на количество пакетов, исходящих от источника. Высокий коэффициент PDR свидетельствует о лучшей производительности протокола.

$$\text{PDR} = \frac{P_r}{P_s} \times 100(\%) \tag{2.1}$$

Где P_r это суммарное количество отправленных пакетов, а P_s это суммарное количество принятых пакетов.

Delay

Среднее время задержки определяется как общее временя от отправления пакета источником до его прибытия к пункту назначения.

$$\text{EED} = \frac{1}{n} \sum_{i=1}^{n} (T_{r_i} - T_{s_i}) \times 1000 \tag{2.2}$$

Где i это идентификатор пакета, n - число успешно доставленных пакетов, T_{r_i} - время приёма пакета, а T_{s_i} это время отправки пакета.

Throughput

Пропускная способность определяется как число пакетов с данными, которые были успешно доставлены в сети за единицу времени. Высокая пропускная способность свидетельствует о лучшей производительности протокола.

$$\text{TH} = \frac{B_r \times 8}{T_{\text{stop}} - T_{\text{start}} \times 1000} \tag{2.3}$$

Где B_r это количество полученных битов, T_{start} - время старта отправки, T_{stop} - время конца отправки.

NRL

Основная цель NRL — измерить количество управляющих данных, необходимых для доставки пакетов данных от источника к получателю, нормализуя это значение на количество успешно доставленных пакетов данных. Это позволяет провести сравнение эффективности маршрутизации между различными протоколами или конфигурациями сети относительно использования сетевых ресурсов.

$$\text{NRL} = \frac{P_{s_{\text{routing}}}}{P_{r_{\text{data}}}} \tag{2.4}$$

Где $P_{s_{\text{routing}}}$ это количество отправленных управляющих пакетов(необходимых для обеспечения маршрутизации), а $P_{r_{\text{data}}}$ - количество успешно принятых пакетов с данными.

Packets sent

Эта метрика отвечает за количество отправленных пакетов в сети и никакой особой формулы для вычисления не имеет, достаточно подсчитывать все отправленные пакеты.

Глава 3

Реализация программного комплекса и анализ результатов эксперимента

3.1 Программа для NS-2

Конфигурация сети

Настройка сети в среде Network Simulator 2 (NS-2) включает в себя установку параметров, которые определяют характеристики симулируемой сети, включая типы устройств, параметры симуляции и конфигурации сетевых протоколов. Эти параметры являются ключевыми элементами для точного моделирования сетевых условий и оценки производительности сети.

Параметры, которые нужно настроить:

Тип канала и модель радиораспространения: Используется беспроводной канал (Channel/WirelessChannel) с моделью двулучевого земного распространения (Propagation/TwoRayGround). Это указывает на физический уровень взаимодействия внутри сети, включая моделирование распространения сигнала и взаимодействия с окружающей средой.

Конфигурация сетевого интерфейса и MAC: Сетевой интерфейс определен как Phy/WirelessPhy, а тип MAC-протокола соответствует стандарту IEEE 802.11 (Mac/802_11). Это отражает распространенные стандарты беспроводной связи при моделировании поведения устройств в сети.

Очередь интерфейса и модель связи: Очередь интерфейса установлена как приоритетная очередь с механизмом DropTail (Queue/DropTail/PriQueue), а для связи между узлами используется тип связи на уровне канала (LL). Это определяет методы управления трафиком и перемещения данных в сети.

Антенна и параметры симуляции: Выбрана всенаправленная антенна

(Antenna/OmniAntenna) для устройств. Параметры среды симуляции включают размер области (956x600 м), максимальное число пакетов в очереди (50) и время окончания симуляции (200 секунд). Это определяет физические и временные рамки моделирования.

Конфигурация через аргументы командной строки: В скрипте предусмотрена возможность изменения некоторых параметров (количество передвижных узлов, входные и выходные файлы, протокол маршрутизации) через аргументы командной строки. Это обеспечивает гибкость и настройку сценария симуляции под конкретные требования.

Динамическая адаптация настроек: В зависимости от выбранного протокола маршрутизации (-rp), тип очереди может быть изменен для оптимизации производительности сети в соответствии с выбранным алгоритмом.

Листинг 3.1: Конфигурация сети в NS-2

```
1  set val(chan)   Channel/WirelessChannel    ;
2  set val(prop)   Propagation/TwoRayGround   ;
3  set val(netif)  Phy/WirelessPhy            ;
4  set val(mac)    Mac/802_11                 ;
5  set val(ifq)    Queue/DropTail/PriQueue    ;
6  set val(ll)     LL                         ;
7  set val(ant)    Antenna/OmniAntenna        ;
8  set val(ifqlen) 50                         ;
9  set val(sn)     6                          ;
10 set val(x)      956                        ;
11 set val(y)      600                        ;
12 set val(stop)   200.0                      ;
13
14 $val(netif) set Pt_ 0.1
15
16 set argc [llength $argv]
17 for {set i 0} {$i < $argc} {incr i} {
18     set arg [lindex $argv $i]
19     if {$arg == "-n"} {
20         incr i
21         set val(mn) [lindex $argv $i]
22         continue
23     }
24     if {$arg == "-f"} {
25         incr i
```

```
26          set val(src) [lindex $argv $i]
27          continue
28      }
29      if {$arg == "-o"} {
30          incr i
31          set val(out) [lindex $argv $i]
32          continue
33      }
34      if {$arg == "-rp"} {
35          incr i
36          set val(rp) [lindex $argv $i]
37          continue
38      }
39  }
40
41  if { $val(rp) == "DSR" } {
42      set val(ifq) CMUPriQueue
43  } else {
44      set val(ifq) Queue/DropTail/PriQueue
45  }
46
47  set val(nn) [expr $val(sn) + $val(mn) ]
```

Далее идёт определение всех необходимых глобальных объектов:

1. Создание объекта симулятора NS-2.
2. Инициализация GOD (General Operations Director).
3. Открытие файла трассировки NS-2.
4. Открытие файла трассировки NAM.
5. Создание беспроводного канала.

Листинг 3.2: Определение глобальных объектов в NS-2

```
1  set ns_ [new Simulator]
2
3  set topo       [new Topography]
4  $topo load_flatgrid $val(x) $val(y)
```

```
5   create-god $val(nn)
6
7   set tracefile [open $val(out)/trace.tr w]
8   $ns_ trace-all $tracefile
9
10  set namfile [open $val(out)/trace.nam w]
11  $ns_ namtrace-all $namfile
12  $ns_ namtrace-all-wireless $namfile $val(x) $val(y)
13  set chan [new $val(chan)];
14
```

Затем необходимо определить параметры мобильных узлов сети.

Проводится настройка параметров мобильного узла с использованием команды node-config, предоставляемой экземпляром симулятора (обычно обозначаемым как $ns_). В рамках данной настройки устанавливаются параметры, такие как алгоритм маршрутизации adhoc (-adhocRouting), типы данных (например: -llType, -macType, -ifqType и т.д.), а также параметры физического слоя (-phyType), канала связи (-channel), и топологии (-topoInstance). Дополнительно активируются опции трассировки (-agentTrace, -routerTrace, -macTrace, -movementTrace), позволяющие следить за различными аспектами симуляции.

Далее в коде следует цикл for, который используется для создания определенного количества узлов в симуляции. Количество создаваемых узлов указывается переменной $val(nn). Для каждого узла создается экземпляр с помощью вызова метода $ns_ node и последующего сохранения полученного узла в массив node_, индексируемый переменной цикла i.

В заключение происходит импорт событий передвижения узлов из файла mobility.tcl.

Листинг 3.3: Настройка узлов сети

```
1 $ns_ node-config -adhocRouting  $val(rp) \
2   -llType         $val(ll) \
3   -macType        $val(mac) \
4   -ifqType        $val(ifq) \
5   -ifqLen         $val(ifqlen) \
6   -antType        $val(ant) \
7   -propType       $val(prop) \
8   -phyType        $val(netif) \
9   -channel        $chan \
10  -topoInstance   $topo \
```

```
11   -agentTrace    ON \
12   -routerTrace   ON \
13   -macTrace      ON \
14   -movementTrace ON
15
16 for {set i 0} {$i < $val(nn) } {incr i} {
17 set node_($i) [$ns_ node]
18 }
19
20 source $val(src)
```

Создание стационарных узлов

Т.к сети VANET предполагают взаимодействие автомобилей со стационарными устройствами, то в рассматриваемом эксперименте надо добавить несколько узлов сети, которые не будут менять свое местоположение. Их необходимо задать вне файла mobility.tcl, в котором определяются все подвижные узлы сети.

Листинг 3.4: Настройка узлов сети

```
1
2 set src_node_i $val(mn)
3 set sink_node_i [expr $src_node_i + 5]
4
5 $node_($src_node_i) set X_ 420
6 $node_($src_node_i) set Y_ 190
7 $node_($src_node_i) set Z_ 0
8
9 $node_([expr $src_node_i + 1]) set X_ 300
10 $node_([expr $src_node_i + 1]) set Y_ 200
11 $node_([expr $src_node_i + 1]) set Z_ 0
12
13 $node_([expr $src_node_i + 2]) set X_ 155
14 $node_([expr $src_node_i + 2]) set Y_ 215
15 $node_([expr $src_node_i + 2]) set Z_ 0
16
17 $node_([expr $src_node_i + 3]) set X_ 130
18 $node_([expr $src_node_i + 3]) set Y_ 345
19 $node_([expr $src_node_i + 3]) set Z_ 0
```

```
20
21 $node_([expr $src_node_i + 4]) set X_ 290
22 $node_([expr $src_node_i + 4]) set Y_ 360
23 $node_([expr $src_node_i + 4]) set Z_ 0
24
25 $node_($sink_node_i) set X_ 115
26 $node_($sink_node_i) set Y_ 490
27 $node_($sink_node_i) set Z_ 0
28
29 for {set i 0} {$i < $val(nn) } {incr i} {
30     $ns_ initial_node_pos $node_($i) 10
31 }
```

Генерация трафика

Финальным этапом настройки сценария симуляции является настройка генерации траффика. В проводимом эксперименте предлагается генерировать трафик из одного стационарного узла в другой. Подвижные же узлы сети будут влиять на маршруты, выбираемые алгоритмами маршрутизации во время проведения симуляции.

Для генерации трафика в NS-2 необходимо назначить узлам TCP агентов

(Agent/TCP/Newreno для источника и Agent/TCPSink для узла назначения) и добавить приложение, которое и будет гегнерировать трафик (Application/FTP).

Листинг 3.5: Генерация траффика

```
1 set tcp_0 [new Agent/TCP/Newreno]
2 $ns_ attach-agent $node_($src_node_i) $tcp_0
3 set sink_0 [new Agent/TCPSink]
4 $ns_ attach-agent $node_($sink_node_i) $sink_0
5 $ns_ connect $tcp_0 $sink_0
6 $tcp_0 set packetSize_ 1500
7
8 set ftp_0 [new Application/FTP]
9 $ftp_0 attach-agent $tcp_0
10 $ns_ at 0.1 "$ftp_0 start"
11 $ns_ at $val(stop) "$ftp_0 stop"
```

Запуск симуляции

После завершения настройки необходимо написать код для запуска симуляции и предусмотреть её корректное завершение.

Листинг 3.6: Запуск сценария и завершение

```
proc finish {} {
  global ns_ tracefile namfile
  $ns_ flush-trace
  close $tracefile
  close $namfile
  exit 0
}
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns_ at $val(stop) "$node_($i) reset"
}
$ns_ at $val(stop) "$ns_ nam-end-wireless $val(stop)"
$ns_ at $val(stop) "finish"
$ns_ at $val(stop) "puts \"done\" ; $ns_ halt"
$ns_ run
```

Когда код программы готов - можно запускать симуляцию через NS-2.

На выходе из симуляции получается трейсфайл с описанием всех событий, происходивших во время симуляции. В нём содержится вся информация о передаваемых пакетах и о том, в какой момент времени в какой узел пакет поступил.

Знание модели позволяет подходящим образом настроить окружение для проведения экспериментов, а так же автоматизировать процесс, где это возможно.

3.2 Настройка окружения

Генерация сценария

Первым этапом моделирования является моделирование мобильности.

Как было описано на схеме моделирования (рис. 2.2), сперва необходимо создать конфигурационный файл для симулятора SUMO. [9]

Этот процесс уже автоматизирован до использования OsmWebWizard.

OsmWebWizard позволяет через графический интерфейс удобно настроить количество автомобилей в эксперименте, силу трафика на дорогах, географическое положение, различные настройки карты, такие как включение/выключение вспомогательных и второстепенных дорог и т.п.

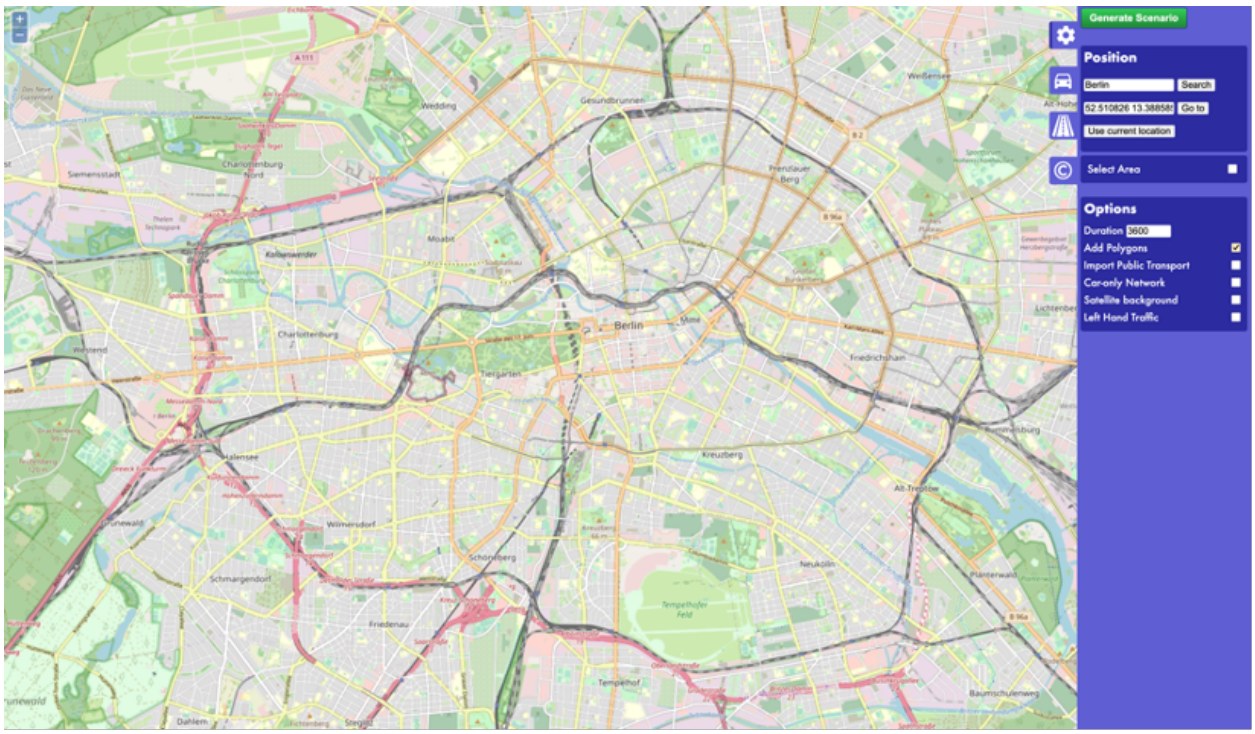

Рис. 3.1: Интерфейс программы OsmWebWizard

Запустить OsmWebWizard можно после простого процесса установки SUMO [11]. Для этого необходимо в директории с установленным симулятором SUMO найти необходимый скрипт на языке Python osmWebWizard.py и запустить его.

Так как эксперимент может проводиться для разного количества узлов (потребуется несколько конфигурационных файлов), то было бы удобно по конфигурационному файлу определить количество подвижных узлов в нём и сразу переименовать его соответствующим образом.

Для этих целей используется вот такой скрипт:

Листинг 3.7: Скрипт для разметки конфиг-файлов по количеству узлов

```
1 #!/bin/bash
2
3 cp -r 2024* tmp_sumo
4 NODES_NUM=$(sumo -c ./tmp_sumo/osm.sumocfg 2>/dev/null
5  | grep Inserted | awk  '{print $2}')
6 echo $NODES_NUM
7 mv tmp_sumo mobility/node_$NODES_NUM
8
9 rm -rf ./2024*
```

Этот скрипт работает очень просто - берётся директория, начинающаяся с 2024(в таком формате osmWebWizard сохраняет результаты своей работы), перекидывается

во временную директорию, а затем выполняется симуляция SUMO. Это нужно, чтобы определить количество узлов, которые будут в итоговой симуляции мобильности.

Примечание: Из интерфейса osmWebWizard не всегда понятно сколько узлов будет после запуска симуляции SUMO, поэтому параметры подбираются под нужное количество узлов.

Итого - получается конфигурационный файл в директории с названием, включающим количество узлов в симуляции по этому файлу. Например если в итоговой симуляции SUMO получилось 100 узлов, то директория с конфигурационным файлом будет называться ‘node_100‘.

Запуск симуляции

Следующим этапом является запуск симуляции из конфигурационного файла, полученного на предыдущем этапе.

Этот процесс можно автоматизировать. Так как конфигурационные файлы уже распределены на предыдущем этапе в директорию mobility/node_*, по количеству узлов, то возможно по всем этим файлам в цикле запустить симуляцию и сохранить трейс-файлы.

Это удобно делать с помощью Makefile.

Листинг 3.8: Makefile для запуска симуляции мобильности

```
1  NAM=../ns-2/ns-allinone-2.36.rc2/nam-1.15/nam
2  NS=../ns-2/ns-allinone-2.36.rc2/ns-2.36/ns
3
4  sumo-trace:
5    @for dir in mobility/*; do \
6      if [ -d "$$dir" ]; then \
7        echo "Creating sumo trace for $$dir"; \
8        (cd $$dir && sumo -c ./osm.sumocfg
9        --fcd-output ./sumoTrace.xml >
10       /dev/null 2>/dev/null); \
11     fi \
12   done
```

Данный makefile сгенерирует по каждому количеству узлов трейс-файл в формате xml.

На самом деле этот этап можно было совместить с предыдущим, но ради изолированности этих процессов, было решено повторно прогонять симуляцию уже с целью вывода в трейс-файл.

Экспорт трейс-файла

Следуя схеме моделирования(рис. 2.2) - следующим этапом эксперимента является преобразование трейс-файла к формату NS-2 [12], это инструмент, в котором проводится моделирование сети VANET.

Итоговый файл должен содержать команды NS-2. Каждая из этих команд будет перемещать узлы сети в соответствии с трейс-файлом мобильности, полученным из SUMO.

Чтобы такой файл получить - нужно запустить скрипт на Python, который так же как и OsmWebWizard поставляется вместе с SUMO - скрипт traceExporter.py

Для полной автоматизации процесса необходимо проходить по всем трейс-файлам в директории mobility и запускать нужный скрипт.

В том же Makefile добавим новую команду.

Листинг 3.9: Makefile для конвертации трейс-файла

```
tcl-trace:
    @for dir in mobility/*; do \
        if [ -d "$$dir" ]; then \
            echo "Exporting NS-2 trace for $$dir"; \
            (cd $$dir &&
      python3 /opt/homebrew/Cellar/sumo/1.19.0/
      share/sumo/tools/traceExporter.py
      --fcd-input sumoTrace.xml
      --ns2mobility-output mobility.tcl
      --shift 10); \
        fi \
    done
```

Автоматизация моделирования сети

Когда автоматизирована мобильность - стоит перейти к автоматизации моделирования сети.

Этот процесс заключается в том, что нужно взять составленный main.tcl [13] файл и запустить его для каждого исследуемого протокола и для каждого количества узлов.

Так же можно добавить новую команду в единый Makefile.

Листинг 3.10: Makefile для запуска симуляции сети

```
ns-trace:
@for protocol in AODV DSDV DSR; do \
```

```
3 for dir in mobility/*; do \
4   if [ -d "$$dir" ]; then \
5     nodes=$$(echo $$dir | grep -o '[0-9]*');\
6     echo "Generating NS-2 result trace
7     for $$protocol with $$nodes cars";\
8     mkdir -p ./net/$$protocol/node_$$nodes 2>/dev/null;\
9     $(NS) net/main.tcl -n $$nodes
10    -f mobility/node_$$nodes/mobility.tcl
11      -o ./net/$$protocol/node_$$nodes -rp $$protocol\
12    echo "NS-2 result trace for $$nodes cars exported
13      to ./net/$$protocol/node_$$nodes";\
14  fi \
15 done \
16 done
```

После выполнения данной команды, получаются трейс-файлы NS-2, которые содержут всю информацию о передаваемых пакетах в сети во время симуляции. Их хорошо использовать для анализа параметров сети.

Вычисление параметров сети

Последним и самым сложным процесс для автоматизации является процесс анализа трейс-файлов NS-2 и рассчёт параметров сети.

В работе рассматриваются пять параметров сети:

1. PDR(Packet Delivery Ratio) - процент доставки пакетов.
2. Delay - время, требующееся для доставки пакета внутри сети.
3. Packets sent - количество отправленных пакетов внутри сети.
4. NRL(Normalized Routing Load) - метрика, показывающая сколько пакетов из всех отправленных служит целям роутинга, а не передаче данных.
5. Throughput - пропускная способность сети.

Зная как устроен трейс-файл NS-2 можно написать необходимый для анализа AWK скрипт [14]. Этот скрипт по сути своей напоминает регулярные выражения, он применяется итеративно ко всему трейс-файлу и в ходе применения может вычислять все необходимые параметры.

В нашем случае для каждого параметра необходимо написать awk скрипт. Примеры таких скриптов можно найти в исходном коде описанного в [15] инструмента.

Когда awk скрипты написаны(см. см. Приложение А), можно писать автоматизацию, которая будет прогонять по этим awk скриптам трейс-файлы с помощью утилиты gawk.

Листинг 3.11: Makefile для подсчёта параметров сети

```
net-params:
@mkdir -p ./results 2>/dev/null;
@params=$$(ls ./awk | sed 's/\.awk$$//' | tr '\n' ','); \
echo "protocol,nodes,$${params%,}" > results/params.csv; \
for protocol in AODV DSDV DSR; do \
  for dir in net/$$protocol/*; do \
    if [ -d "$$dir" ]; then \
    nodes=$$(echo $$dir | grep -o '[0-9]*'); \
    echo "Computing $$protocol for $$nodes nodes"; \
    tracefile="net/$$protocol/node_$$nodes/trace.tr"; \
    results="$$protocol,$$nodes"; \
    for awkscript in ./awk/*.awk; do \
      paramname=$$(basename -s .awk $$awkscript); \
      paramvalue=$$(gawk -f $$awkscript $$tracefile); \
      results+=",${results}$$paramvalue"; \
    done; \
    echo "$$results" >> results/params.csv; \
    fi \
  done \
done;
```

По сути эта команда берёт все awk скрипты из директории awk и прогоняет по ним все имеющиеся трейс-файлы NS-2.

Каждый awk скрипт служит для извлечения одного параметра сети, в качестве имени параметра служит имя файла с awk скриптом. Далее каждый из параметров записывается в итоговую таблицу формата CSV.

Таким образом, если потребуется добавить новый параметр - всего лишь нужно будет добавить awk скрипт в директорию awk и запустить команду net-params в Makefile.

Полученный результат отображён в Таблице 3.1.

protocol	nodes	delay	nrl	packets_sent	pdr	throughput
AODV	100	186.337	14.1258	5631	84.6741	152.431
AODV	20	200.137	4.1225	5267	80.5202	145.692
AODV	40	183.867	6.4962	5725	82.2009	157.59
AODV	60	168.348	9.91878	5609	83.3482	149.298
AODV	80	174.738	12.3182	5568	83.0999	152.56
DSDV	100	220.073	13.7952	23959	10.0004	72.4146
DSDV	20	321	2.39105	4759	38.7056	99.2031
DSDV	40	212.081	3.06161	9308	32.9824	103.884
DSDV	60	224.664	4.94847	13079	23.6868	97.7795
DSDV	80	220.798	11.0758	15694	12.3805	67.7485
DSR	100	579.803	1.07585	15742	42.1675	194.707
DSR	20	733.798	0.698134	25257	21.7207	165.868
DSR	40	678.565	0.646085	16621	41.7725	202.047
DSR	60	663.989	0.848163	16790	40.4348	197.62
DSR	80	613.153	1.02997	14569	42.8169	183.562

Таблица 3.1: Таблица с результатами эксперимента

Построение графиков

Последним этапом является автоматизация построения графиков для получившейся таблицы.

Для этого будет удобно написать python скрипт, который считает таблицу с параметрами в формате CSV и нарисует все необходимые графики.

Листинг 3.12: Python скрипт для построения графиков

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3
4  df = pd.read_csv('./results/params.csv')
5
6  parameters = df.columns.tolist()[2:]
7
8  protocols = df['protocol'].unique()
9
10 markers = ['o', 's', '^']
11 lines = ['-', '--', '-.']
12
13 protocol_styles = {protocol: {'marker': markers[i], 'line':
   lines[i]} for i, protocol in enumerate(protocols)}
14
```

```
15     for param in parameters:
16         plt.figure(figsize=(10, 6))
17         for protocol in protocols:
18             protocol_df = df[df['protocol'] == protocol]
19             protocol_df = protocol_df.sort_values(by='nodes')
20 
21             style = protocol_styles[protocol]
22             plt.plot(protocol_df['nodes'], protocol_df[param],
    marker=style['marker'], linestyle=style['line'], label=
    protocol)
23 
24         plt.title(f'{param.upper()} vs количество узлов')
25         plt.xlabel('Количество узлов')
26         plt.ylabel(param.upper())
27         plt.legend()
28         plt.grid(True)
29 
30         plt.savefig(f'plots/{param}.png')
```

После запуска этого скрипта получаются графики по каждому из параметров сети. На каждом из графиков отображена зависимость значения параметра сети от количества подвижных узлов в сети для каждого протокола маршрутизации. [16]

3.3 Анализ результатов

Возникшие проблемы

В процессе неоднократных попыток запуска эксперимента протоколы DSR и AODV стабильно показывали похожий на реальные значения результат, а именно - отличные от нуля параметры сети, такие как количество отправленных пакетов, процент доставки пакетов, задержка и тд. Но была одна проблема - DSDV не работал в поставленных условиях симуляции.

Это проявлялось в том, что при запуске симуляции с использованием протокола DSDV, стационарные узлы не успевали начать отправлять пакеты. Возникло предположение, что это происходит из-за постоянно и быстро меняющейся топологии сети. Конкретно - узлы двигались с большой скоростью из-за чего не успевала построиться таблица маршрутизации, необходимая для работы DSDV.

Решением данной проблемы стало добавление вот новых строк в код программы,

производящих настройку агента DSDV.

По сути эти настройки увеличивают производительность и частоту построения таблицы DSDV.

Необходимо пояснить эти настройки.

Строка ‘Agent/DSDV set perup_ 6 ;‘

Эта строка устанавливает значение параметра perup_ для агента DSDV равным 6. Параметр perup_ относится к периодичности (времени в секундах), с которой DSDV генерирует полные объявления маршрутизации. В данном случае, полные объявления будут генерироваться каждые 6 секунд. Это позволяет узлам обновлять информацию о маршрутизации в своих таблицах маршрутизации с указанной периодичностью.

Строка ‘Agent/DSDV set use_mac_ 0 ;‘

Здесь устанавливается параметр use_mac_ равным 0 для агента DSDV. Параметр use_mac_ определяет, будет ли для передачи сообщений объявлений DSDV использоваться MAC (Media Access Control) адресация. Значение 0 указывает, что MAC адресация не будет использоваться. В контексте симуляции это может быть полезно для упрощения модели или когда не требуется моделирование на уровне доступа к среде передачи данных. По сути это сильно улучшает производительность и ускоряет симуляцию.

Строка ‘Agent/DSDV set min_update_periods_ 2 ;‘

Эта строка устанавливает минимальное количество периодов между отправками полных объявлений, параметр min_update_periods_, равным 2 для агента DSDV. Это значит, что между consecutive (последовательными) полными объявлениями должно пройти не менее двух периодов выпуска этих объявлений. Это может быть использовано для ограничения количества сетевого трафика, вызванного передачей объявлений, путем установления минимального интервала времени между объявлениями.

Листинг 3.13: Дополнительные настройки для Agent/DSDV

```
1 if { $val(rp) == "DSDV" } {
2     Agent/DSDV set perup_           6        ;
3     Agent/DSDV set use_mac_         0        ;
4     Agent/DSDV set min_update_periods_ 2     ;
5 }
```

Кроме того, для уменьшения скорости передвижения узлов была использована программа NetEdit. Она позволяет задать ограничение для симуляции мобильности и отредактировать "карту местности" которая будет отправлена в SUMO. С помощью этого инструмента была уменьшена максимальная скорость на рассматриваемом участке местности, чтобы увеличить время нахождения узлов в сети и расширить окно, в котором существует способ передавать данные между двумя узлами.

Сравнение протоколов

После всех правок все три протокола стали давать ожидаемые метрики и визуально(через NetAnim) стало возможно наблюдать передачу данных в момент, когда подвижные узлы создают сеть, по которой становится возможно передать данные из одного стационарного узла в другой.

Настроенное окружение позволяет сразу создать и проанализировать графики зависимости параметров сети от количества узлов в сети для всех рассматриваемых протоколов маршрутизации.

Параметры сети были выбраны следующие:

1. PDR(Packet Delivery Ratio) - процент доставки пакетов.
2. Delay - время, требующееся для доставки пакета внутри сети.
3. Packets sent - количество отправленных пакетов внутри сети.
4. NRL(Normalized Routing Load) - метрика, показывающая сколько пакетов из всех отправленных служит целям роутинга, а не передаче данных.
5. Throughput - пропускная способность сети.

Метрика End-to-end delay

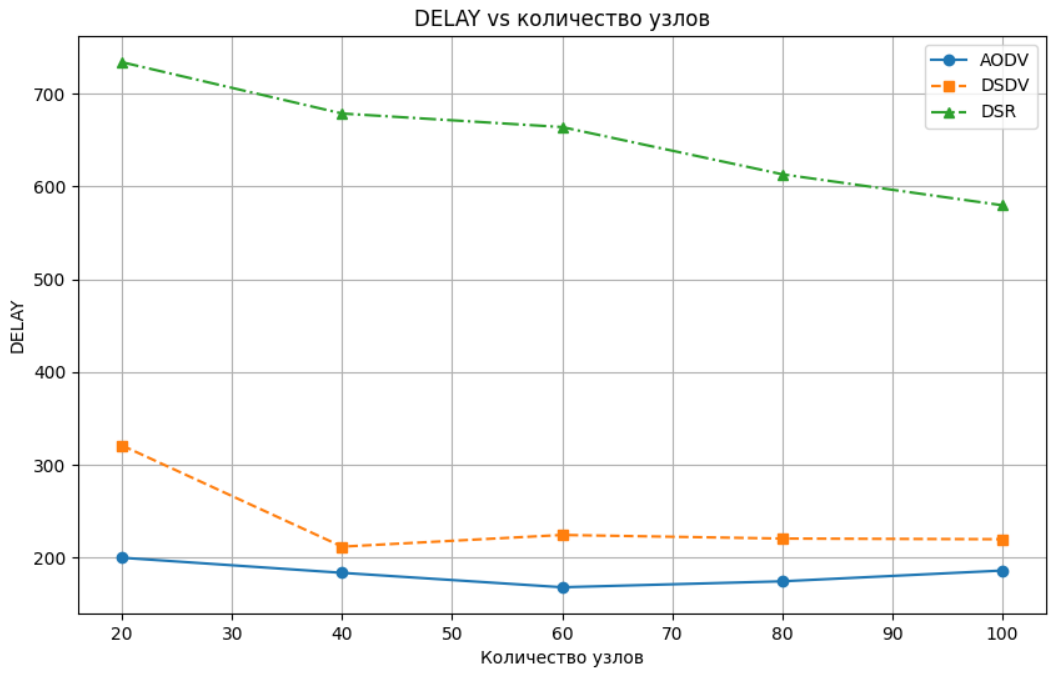

Рис. 3.2: Параметр сети Delay

По рис. 3.2 видно, что лучше всех показатель задержки у протокола AODV и в зависимости от количества узлов в сети положение вещей не меняется. Второе место занимает DSDV, он не так сильно отстаёт от AODV, но в поставленном эксперименте немного проигрывает. Протокол DSR в параметре задержки сильно уступает двум другим протоколам. В зависимости от количества узлов в сети задержка для протокола DSR уменьшается довольно сильно, возможно играет роль плотность узлов в сети.

Метрика NRL

Исходя из рис. 3.3, параметр NRL закономерно выше у AODV и DSDV, т.к достаточно большое количество пакетов отправленных в сеть служат только для нахождения необходимого пути и построения таблиц маршрутизации. Для DSR такой проблемы нет, что обсуловлено устройством DSR. Видно, что AODV и DSDV сильно не отличаются по этому параметру, а DSR заметно и кратно выигрывает. Так же понятно и подтверждается, что при увеличении количества узлов - процент пакетов, отправленных для целей маршрутизации в протоколах DSDV и AODV сильно увеличивается, а у DSR остаётся примерно на прежнем уровне, лишь показывая очень слабый, но закономерный рост.

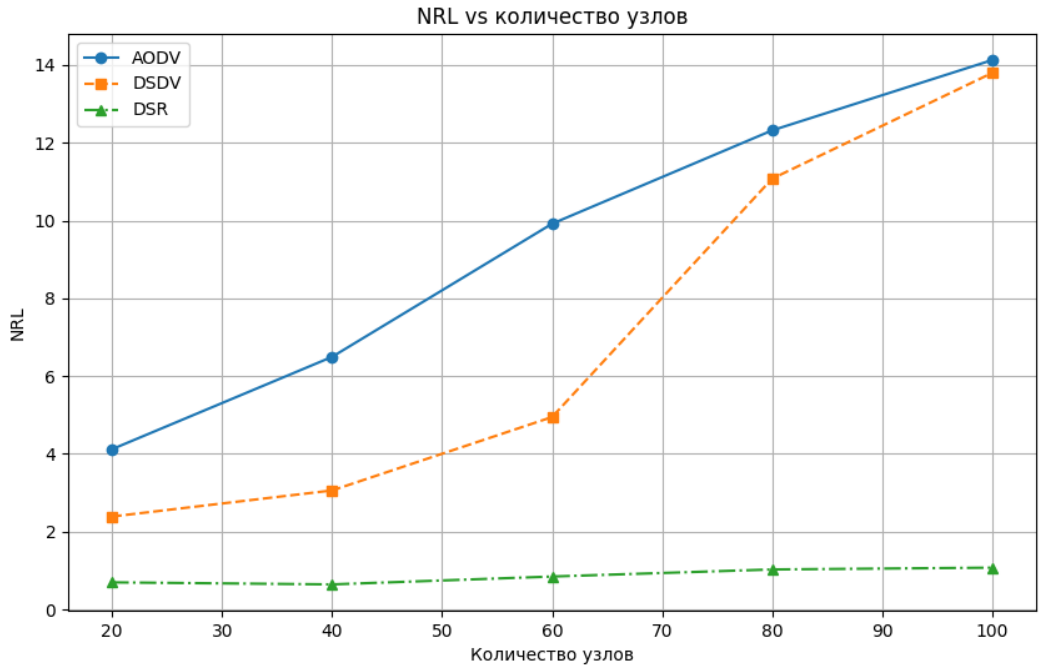

Рис. 3.3: Параметр сети NRL

Метрика Packets sent

Исходя из рис. 3.4, количество отправленных пакетов в сеть у протоколов DSR и DSDV заметно выше чем у протокола AODV. Видно, что у протокола DSDV при увеличении

количества узлов количество отправленных в сеть пакетов сильно возрастает, когда у протокола DSR после определённого количества узлов остаётся примерно неизменной. У протокола AODV в целом от количества узлов сильно не меняется количество отправленных пакетов и остаётся на стабильно низком уровне.

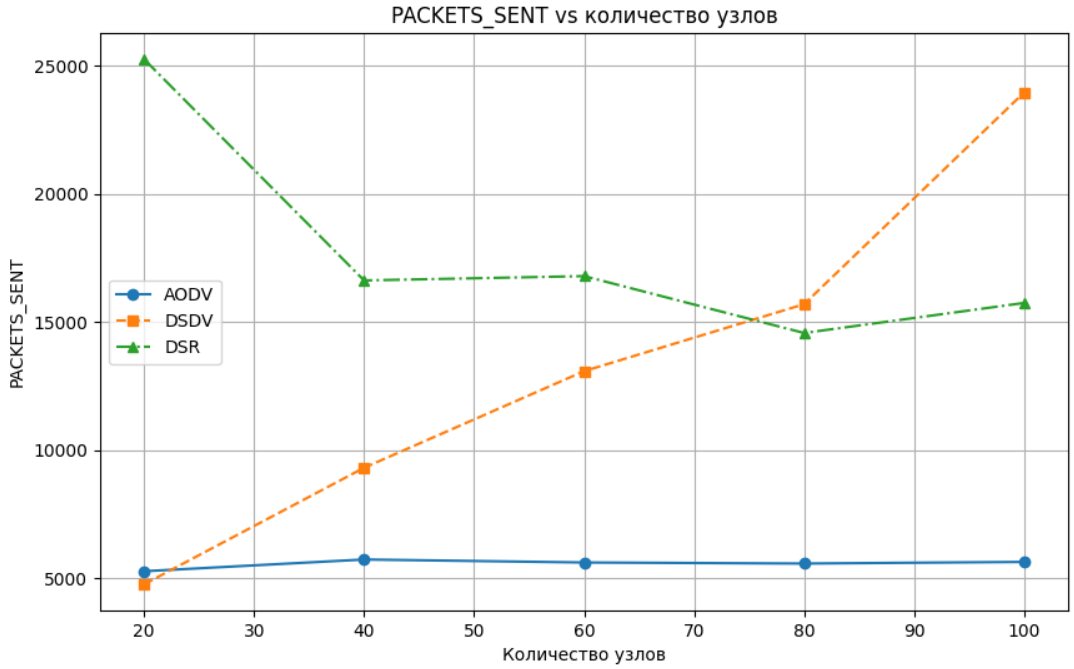

Рис. 3.4: Параметр сети Packets sent

Метрика PDR

Исходя из рис. 3.5, процент доставки пакетов заметно лучше у протокола AODV. Он начинается от 80% и стабильно растёт при увеличении количества узлов в сети. У протокола DSR ситуация по тенденции роста похожая, но рост не так очевиден и в целом средний уровень процента доставки пакетов ниже примерно в два раза чем у AODV. Интересная ситуация происходит с протоколом DSDV. При увеличении количества узлов этот протокол показывает очень плохие результаты по доставке пакетов, вплоть до 10%, что конечно по сравнению с AODV и DSR очень плохо.

Метрика Throughput

Исходя из рис. 3.6 пропускная способность самая большая оказалась у DSR и составляет в среднем больше 180 kbps. На втором месте протокол AODV со значением в районе 150 kbps. Протокол DSDV в параметре пропускной способности опять оказывается самым худшим из трёх. Пропускная способность при использовании DSDV опускается

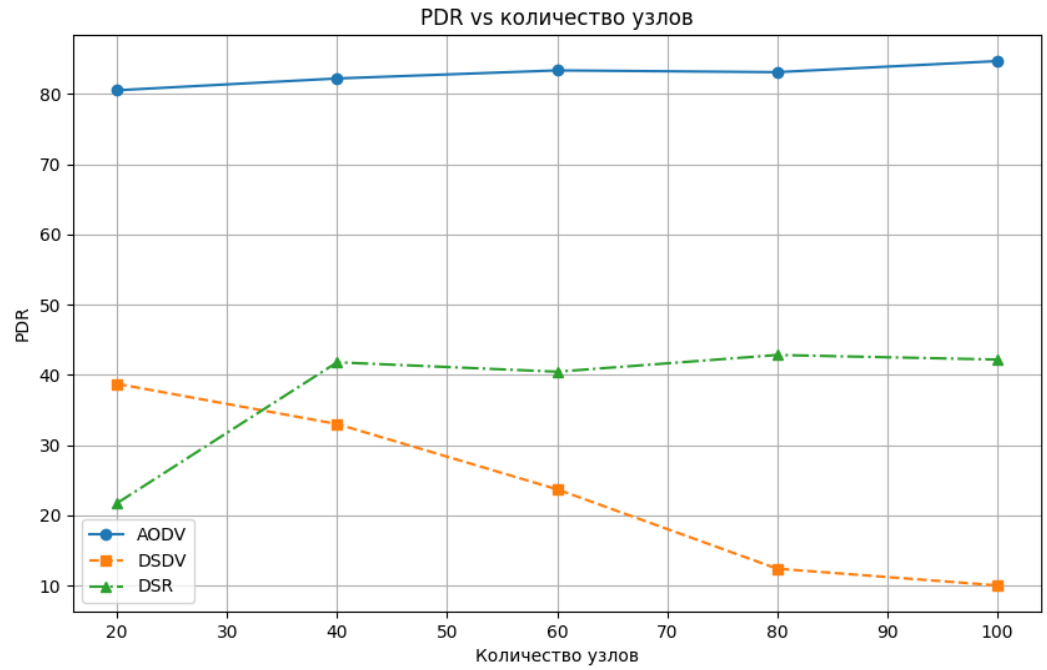

Рис. 3.5: Параметр сети PDR

ниже 80 kbps при увеличении количества узлов. Кроме того, можно заметить, что у протоколов DSR и AODV практически не изменяется пропускная способность от изменения количества узлов и примерно остаётся на одном уровне, в то время как при использовании DSDV количество узлов влияет на пропускную способность.

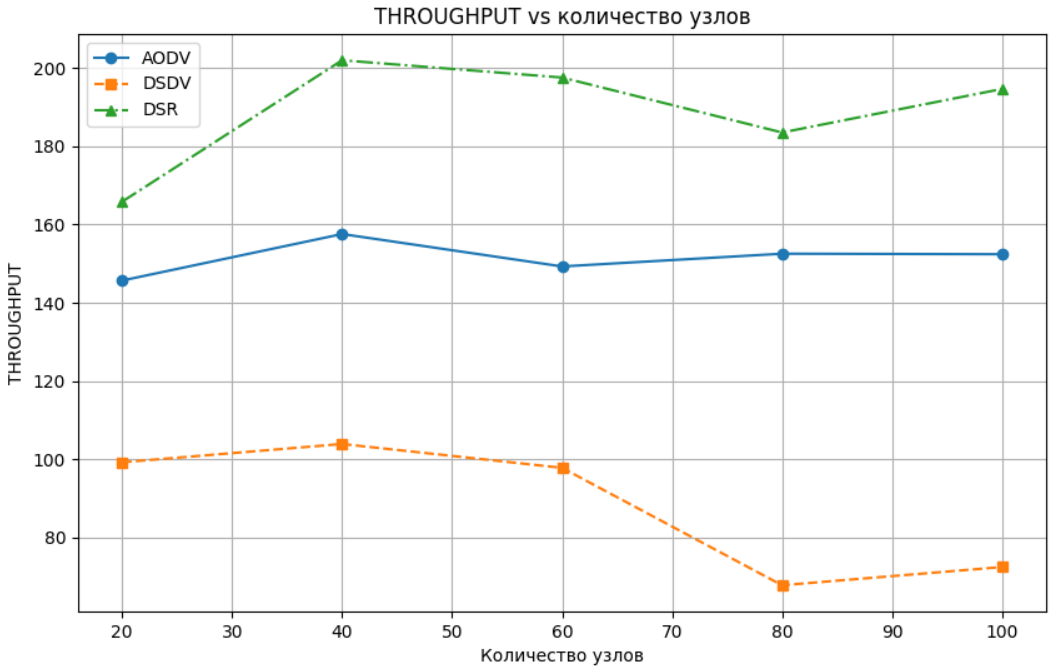

Рис. 3.6: Параметр сети Throughput

На основе проведенного анализа результатов эксперимента, относящегося к параметрам сети таким как PDR (Packet Delivery Ratio), Delay (время доставки пакета), Packets Sent (количество отправленных пакетов), NRL (Normalized Routing Load), и Throughput (пропускная способность сети), можно сделать следующие выводы:

1. Анализ метрики Delay показывает, что протокол AODV демонстрирует наилучшие показатели задержки, что сохраняется независимо от количества узлов в сети. Протоколы DSDV и DSR показывают более высокие значения задержки, с относительным улучшением показателей DSR при увеличении плотности сети.

2. Исследование метрики NRL выявляет, что DSR значительно превосходит AODV и DSDV по эффективности использования сетевых ресурсов, продемонстрировав более низкий уровень роутинговой нагрузки. Это указывает на меньшее количество контрольных пакетов, необходимых для маршрутизации в протоколе DSR.

3. Показатель общего количества отправленных пакетов (Packets sent) подчеркивает эффективность AODV в поддержании низкого уровня сетевого трафика, в то время как DSDV и DSR испытывают значительное увеличение числа отправляемых пакетов при росте сети, что особенно выражено у DSDV.

4. Анализ метрики PDR демонстрирует, что AODV обеспечивает наивысший процент успешной доставки пакетов, что делает его предпочтительным выбором для обеспечения надежности сетевой передачи данных. DSR и DSDV показывают более низкие результаты, причем DSDV выявил значительное падение производительности при увеличении числа узлов.

5. Оценка пропускной способности сети (Throughput) показывает, что, несмотря на наивысшие показатели у DSR, AODV обеспечивает сбалансированное сочетание производительности и надежности сети, что подчеркивается его хорошими результатами и в метрике Throughput, и в других рассмотренных параметрах.

Итак, учитывая общую производительность в контексте рассмотренных метрик, протокол AODV демонстрирует себя как наиболее подходящий для использования в условиях проведенного эксперимента. Это обусловлено его способностью обеспечивать высокую пропускную способность, низкую задержку, а также высокий PDR, что в совокупности подтверждает его эффективность для обеспечения качественной и надежной сетевой коммуникации. Данный вывод коррелирует с результатами, полученными в исследовании, упомянутом в [17], подчеркивая превосходство AODV над другими рассмотренными протоколами в разнообразных сценариях использования.

В [17] авторы обозревают и анализируют различные протоколы маршрутизации, чтобы выяснить, какой из них наилучшим образом подходит для применения в VANET,

особенно для задач связанных с видео потоковой передачей. Основной вывод работы заключается в том, что протокол AODV показывает наилучшие результаты как в стандартных, так и в более сложных сценариях использования VANET, включая передачу видео, по сравнению с другими адаптивными сетевыми протоколами, такими как DSR. Как можно наблюдать по результатам нашего эксперимента - протокол AODV так же даёт лучший результат почти что по всем параметрам относительно других протоколов.

Заключение

В ходе выполнения выпускной квалификационной работы бакалавра достигнуты поставленные цели и решены ключевые задачи, направленные на исследование и анализ эффективности протоколов маршрутизации в сетях VANET с использованием инструментов моделирования SUMO и NS-2. Было проведено подробное сравнение трех протоколов маршрутизации: DSDV, AODV и DSR, с целью выявления их эффективности в условиях, когда стационарные узлы взаимодействуют с использованием подвижных узлов, формирующих динамичную автомобильную сеть.

Основные задачи, успешно реализованные в рамках данной работы, включали в себя:

1. Составление детального описания имитационной модели, предназначенной для анализа протоколов маршрутизации в сетях VANET. Это обеспечило необходимую основу для дальнейшего исследования и анализа.

2. Разработка программного комплекса, способствующего проведению экспериментов с сетями VANET в разнообразных условиях. Разработанный комплекс оказал значительную поддержку в анализе и сравнении протоколов маршрутизации, что является ключевой целью работы.

3. Выполнение аналитического анализа результатов экспериментов. В результате анализа были получены выводы относительно эффективности исследуемых протоколов маршрутизации, что способствовало глубокому пониманию работы и особенностей сетей VANET в представленных условиях.

Выполненная работа продемонстрировала анализ выбранных протоколов маршрутизации, их практическое применение в условиях сетей VANET и способность адаптации к специфическим условиям этих сетей. Результаты исследования могут быть использованы в качестве основы для дальнейшего развития исследований в области интеллектуальных транспортных систем и сетей VANET, способствуя развитию эффективных методов маршрутизации и повышению качества коммуникационных процессов в автомобильных сетях.

Таким образом, достигнутые результаты подтверждают актуальность и значимость исследованной проблематики, важность расширения знаний и разработки новых подходов в области моделирования и анализа сетей VANET.

Приложение А

Файлы awk

Листинг А.1: AWK скрипт для подсчёта параметра delay

```
1   BEGIN {
2     seqno = -1;
3     count = 0;
4   }{
5     if($4 == "AGT" && $1 == "s" && seqno < $6) {
6           seqno = $6;
7     }
8     if($4 == "AGT" && $1 == "s") {
9           start_time[$6] = $2;
10    } else if(($7 == "tcp") && ($1 == "r")) {
11        end_time[$6] = $2;
12    } else if($1 == "D" && $7 == "tcp") {
13          end_time[$6] = -1;
14    }
15  } END {
16    for (i=0; i<=seqno; i++) {
17        if(end_time[i] > 0) {
18            delay[i] = end_time[i] - start_time[i];
19            count++;
20        } else {
21            delay[i] = -1;
22        }
23    }
24    for (i=0; i<=seqno; i++) {
25        if(delay[i] > 0) {
26            n_to_n_delay = n_to_n_delay + delay[i];
```

```
27          }
28      }
29      n_to_n_delay = n_to_n_delay/count;
30      print n_to_n_delay * 1000;
31    }
32
```

Листинг А.2: AWK скрипт для подсчёта параметра NRL

```
1   BEGIN {
2     recvd = 0;
3     rt_pkts = 0;
4   }{
5     if (( $1 == "r") && ( $7 == "cbr" || $7 =="tcp" ) && ( $4
    =="AGT" ))
6     recvd++;
7     if (($1 == "s" || $1 == "f") && $4 == "RTR" && ($7 ==""
    AODV" || $7
8     =="message" || $7 =="DSR" || $7 =="OLSR" || $7 == "DSDV")
    ) rt_pkts++;
9   }END {
10    print rt_pkts/recvd;
11  }
12
```

Листинг А.3: AWK скрипт для подсчёта параметра Packets sent

```
1     BEGIN {
2     seqno = -1;
3   }{
4     if ($4 == "AGT" && $1 == "s" && seqno < $6) {
5       seqno = $6;
6     }
7   } END {
8     print seqno+1;
9   }
10
```

Листинг А.4: AWK скрипт для подсчёта параметра PDR

```
1     BEGIN {
2       seqno = -1;
3       droppedPackets = 0;
4       receivedPackets = 0;
5       count = 0;
6     } {
7       event = $1
8       time = $2
9       pkt_size = $8
10      level = $4
11      if(level == "AGT" && event == "s" && seqno < $6) {
12        seqno = $6;
13      } else if((level == "AGT") && (event == "r")) {
14        receivedPackets++;
15      } else if (event == "D" && $7 == "tcp" && pkt_size > 512)
    {
16        droppedPackets++;
17      }
18    } END {
19      print receivedPackets/(seqno+1)*100;
20    }
21
```

Листинг A.5: AWK скрипт для подсчёта параметра Throughput

```
1     BEGIN {
2       recvdSize = 0
3       startTime = 400
4       stopTime = 0
5     } {
6       event = $1
7       time = $2
8       node_id = $3
9       pkt_size = $8
10      level = $4
11      if (level == "AGT" && event == "s" && pkt_size >= 512) {
12        if (time < startTime) {
13          startTime = time
```

```
14          }
15        }
16        if (level == "AGT" && event == "r" && pkt_size >= 512) {
17          if (time > stopTime) {
18            stopTime = time
19          }
20          hdr_size = pkt_size % 512
21          pkt_size -= hdr_size
22          recvdSize += pkt_size
23        }
24      } END {
25        print (recvdSize/(stopTime-startTime))*(8/1000)
26      }
27
```

Приложение В

Файл main.tcl

Листинг В.1: Файл main.tcl

```
1
2 set val(chan)   Channel/WirelessChannel    ;
3 set val(prop)   Propagation/TwoRayGround   ;
4 set val(netif)  Phy/WirelessPhy            ;
5 set val(mac)    Mac/802_11                 ;
6 set val(ifq)    Queue/DropTail/PriQueue    ;
7 set val(ll)     LL                         ;
8 set val(ant)    Antenna/OmniAntenna        ;
9 set val(ifqlen) 50                         ;
10 set val(sn)     3                         ;
11 set val(x)      1500                      ;
12 set val(y)      1500                      ;
13 set val(stop)   500.0                     ;
14
15 $val(netif) set Pt_ 0.3
16
17 set argc [llength $argv]
18 for {set i 0} {$i < $argc} {incr i} {
19     set arg [lindex $argv $i]
20     if {$arg == "-n"} {
21        incr i
22        set val(mn) [lindex $argv $i]
23        continue
24     }
25     if {$arg == "-f"} {
26        incr i
```

```
27         set val(src) [lindex $argv $i]
28         continue
29     }
30     if {$arg == "-o"} {
31         incr i
32         set val(out) [lindex $argv $i]
33         continue
34     }
35     if {$arg == "-rp"} {
36         incr i
37         set val(rp) [lindex $argv $i]
38         continue
39     }
40 }
41
42 if { $val(rp) == "DSDV" } {
43     Agent/DSDV set perup_           6        ;
44     Agent/DSDV set use_mac_         0        ;
45     Agent/DSDV set min_update_periods_ 2     ;
46 }
47
48 if { $val(rp) == "DSR" } {
49     set val(ifq) CMUPriQueue
50 } else {
51     set val(ifq) Queue/DropTail/PriQueue
52 }
53
54 set val(nn) [expr $val(sn) + $val(mn) ]
55
56 set ns_ [new Simulator]
57
58 set topo       [new Topography]
59 $topo load_flatgrid $val(x) $val(y)
60 create-god $val(nn)
61
62 set tracefile [open $val(out)/trace.tr w]
63 $ns_ trace-all $tracefile
```

```
64
65 set namfile [open $val(out)/trace.nam w]
66 $ns_ namtrace-all $namfile
67 $ns_ namtrace-all-wireless $namfile $val(x) $val(y)
68 set chan [new $val(chan)];
69
70 $ns_ node-config -adhocRouting  $val(rp) \
71          -llType        $val(ll) \
72          -macType       $val(mac) \
73          -ifqType       $val(ifq) \
74          -ifqLen        $val(ifqlen) \
75          -antType       $val(ant) \
76          -propType      $val(prop) \
77          -phyType       $val(netif) \
78          -channel       $chan \
79          -topoInstance  $topo \
80          -agentTrace    ON \
81          -routerTrace   ON \
82          -macTrace      ON \
83          -movementTrace ON
84
85
86 for {set i 0} {$i < $val(nn) } {incr i} {
87     set node_($i) [$ns_ node]
88 }
89
90 source $val(src)
91
92 set src_node_i $val(mn)
93 set sink_node_i [expr $src_node_i + 2]
94
95 $node_($src_node_i) set X_ 300
96 $node_($src_node_i) set Y_ 785
97 $node_($src_node_i) set Z_ 0
98
99 $node_([expr $src_node_i + 1]) set X_ 525
100 $node_([expr $src_node_i + 1]) set Y_ 605
```

```
$node_([expr $src_node_i + 1]) set Z_ 0

$node_($sink_node_i) set X_ 750
$node_($sink_node_i) set Y_ 425
$node_($sink_node_i) set Z_ 0

for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ initial_node_pos $node_($i) 10
}

set tcp_0 [new Agent/TCP/Newreno]
$ns_ attach-agent $node_($src_node_i) $tcp_0
set sink_0 [new Agent/TCPSink]
$ns_ attach-agent $node_($sink_node_i) $sink_0
$ns_ connect $tcp_0 $sink_0
$tcp_0 set packetSize_ 1500

set ftp_0 [new Application/FTP]
$ftp_0 attach-agent $tcp_0
$ns_ at 0.1 "$ftp_0 start"
$ns_ at $val(stop) "$ftp_0 stop"

proc finish {} {
    global ns_ tracefile namfile
    $ns_ flush-trace
    close $tracefile
    close $namfile
    exit 0
}
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns_ at $val(stop) "$node_($i) reset"
}
$ns_ at $val(stop) "$ns_ nam-end-wireless $val(stop)"
$ns_ at $val(stop) "finish"
$ns_ at $val(stop) "puts \"done\" ; $ns_ halt"
$ns_ run
```

Приложение C

Файл Makefile

Листинг C.1: Файл для автоматизации запуска эксперимента Makefile

```
1 NAM=../ns-2/ns-allinone-2.36.rc2/nam-1.15/nam
2 NS=../ns-2/ns-allinone-2.36.rc2/ns-2.36/ns
3
4 .PHONY: all
5 all: sumo-trace tcl-trace ns-trace
6
7 sumo-trace:
8     @for dir in mobility/*; do \
9         if [ -d "$$dir" ]; then \
10             echo "Creating sumo trace for $$dir"; \
11             (cd $$dir && sumo -c ./osm.sumocfg  --fcd-output ./
    sumoTrace.xml > /dev/null 2>/dev/null); \
12         fi \
13     done
14
15 tcl-trace:
16     @for dir in mobility/*; do \
17         if [ -d "$$dir" ]; then \
18             echo "Exporting NS-2 trace for $$dir"; \
19             (cd $$dir && python3 /opt/homebrew/Cellar/sumo
    /1.19.0/share/sumo/tools/traceExporter.py --fcd-input
    sumoTrace.xml --ns2mobility-output mobility.tcl --shift 10);
     \
20         fi \
21     done
22
```

```
23 ns-trace:
24     @for protocol in AODV DSDV DSR; do \
25         for dir in mobility/*; do \
26             if [ -d "$$dir" ]; then \
27                 nodes=$$(echo $$dir | grep -o '[0-9]*');\
28                 echo "Generating NS-2 result trace for
    $$protocol with $$nodes cars";\
29                 mkdir -p ./net/$$protocol/node_$$nodes 2>/dev/
    null;\
30                 $(NS) net/main.tcl -n $$nodes -f mobility/
    node_$$nodes/mobility.tcl -o ./net/$$protocol/node_$$nodes -
    rp $$protocol\
31                 echo "NS-2 result trace for $$nodes cars
    exported to ./net/$$protocol/node_$$nodes";\
32             fi \
33         done \
34     done
35
36 net-params:
37     @mkdir -p ./results 2>/dev/null;
38     @params=$$(ls ./awk | sed 's/\.awk$$//' | tr '\n' ','); \
39     echo "protocol,nodes,$${params%,}" > results/params.csv; \
40     for protocol in AODV DSDV DSR; do \
41         for dir in net/$$protocol/*; do \
42             if [ -d "$$dir" ]; then \
43                 nodes=$$(echo $$dir | grep -o '[0-9]*'); \
44                 echo "Computing $$protocol for $$nodes nodes";
    \
45                 tracefile="net/$$protocol/node_$$nodes/trace.tr
    "; \
46                 results="$$protocol,$$nodes"; \
47                 for awkscript in ./awk/*.awk; do \
48                     paramname=$$(basename -s .awk $$awkscript);
     \
49                     paramvalue=$$(gawk -f $$awkscript
    $$tracefile); \
50                     results+=",${results}$$paramvalue"; \
```

```
51                  done; \
52                  echo "$$results" >> results/params.csv; \
53              fi \
54          done \
55      done;
56
57 params-plots:
58      @./plots/exp_venv/bin/python3.12 ./plots/plots.py
```

Литература

[1] Qureshi K. N., Abdullah A. H. A survey on intelligent transportation systems // Middle-East Journal of Scientific Research. 2013. Vol. 15. No. 5. P. 629–642.

[2] Anwer M. S., Guy C. A survey of vanet technologies // Journal of Emerging Trends in Computing and Information Sciences. 2014. Vol. 5. No. 9. P. 661–671.

[3] Hahn D., Munir A., Behzadan V. Security and privacy issues in intelligent transportation systems: Classification and challenges // IEEE Intelligent Transportation Systems Magazine. 2019. Vol. 13. No. 1. P. 181–196.

[4] Analysis of position based routing vanet protocols using ns2 simulator / Pothuganti Karunakar, Jagadish Matta, RP Singh, O Ravi Kumar // International Journal of Innovative Technology and Exploring Engineering (IJITEE). 2020. Vol. 9. No. 5. P. 01–04.

[5] Pande S., Sadakale R., Ramesh N. Performance analysis of aodv routing protocol in vanet using ns-2 and sumo // WCNC-2021: Workshop on Computer Networks & Communications. 2021.

[6] Kathiriya H., Bavarva A. Traffic detection in vanet using ns2 and sumo // Traffic. 2013. P. 1–7.

[7] Simulation of vanet using ns-3 and sumo / Chitraxi Raj, Urvik Upadhayaya, Twinkle Makwana, Payal Mahida // International Journal of Advanced Research in Computer Science and Software Engineering. 2014. Vol. 4. No. 4.

[8] Hassan A. Vanet simulation. 2009.

[9] Simulation-based performance evaluation of vanet routing protocols under indian traffic scenarios / Sudesh Kumar Seema, Suresh Kumar Sharma, Suhel Ahmad Khan, Pawan Singh // ICIC Express Letters. 2022. Vol. 16. No. 1. P. 67–74.

[10] Mahdi H. F., Abood M. S., Hamdi M. M. Performance evaluation for vehicular ad-hoc networks based routing protocols // Bulletin of Electrical Engineering and Informatics. 2021. Vol. 10. No. 2. P. 1080–1091.

[11] Документация SUMO. 2001. Access mode: https://sumo.dlr.de/docs/.

[12] Network Simulator 2. 2000. Access mode: https://ns2simulator.com/ns2-download/.

[13] Примеры скриптов ns-2. 2021. Access mode: https://github.com/fayazur-mohammad/DCN-with-ns-2/blob/main/Exp02b.tcl.

[14] Скрипты для анализа трейс-файлов NS-2. 2013. Access mode: https://www.nsnam.com/2013/03/awk-scripts-for-ns2-to-process-data.html.

[15] Инструменты для анализа трейс-файлов NS-2. 2014. Access mode: https://www.nsnam.com/2014/11/app-tool-for-analysing-tracefiles-for.html.

[16] Sarao P. Comparison of aodv, dsr, and dsdv routing protocols in a wireless network. // J. Commun. 2018. Vol. 13. No. 4. P. 175–181.

[17] Rizwan Ghori M., Safa Sadiq A., Ghani A. Vanet routing protocols: review, implementation and analysis // Journal of physics: conference series / IOP Publishing. Vol. 1049. 2018. P. 012064.