

Министерство Высшего образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого

—
Институт прикладной математики и механики
Высшая Школа Кибербезопасности и Защиты Информации

КУРСОВАЯ РАБОТА

«Сравнительный анализ эмуляторов ad-hoc сетей»

по дисциплине «Модели безопасности компьютерных систем»

Выполнил
студент гр. 3651003/70801

<подпись>

Гасанов Э.А.

Проверил
ассистент

<подпись>

Овасапян Т. Д.

Санкт-Петербург
2020

Содержание

Введение	3
Цель работы	3
Задачи	3
1.Хронология сетевых симуляторов.....	4
2.Архитектура симуляторов NS-2 и NS-3.....	6
2.1 Архитектура NS-2	6
2.1.1 Основные понятия в NS-2	9
2.1.2 Планировщик событий в NS-2.....	9
2.1.3 Абстракции транспортного и прикладного уровней: Агенты и Приложения	10
2.1.4 Описание trace файла в NS-2	11
2.1.5 Новый формат трассировочного файла	12
2.1.6 Старый формат трассировочного файла.....	13
2.2 Архитектура NS-3	14
2.2.1 Трассировочный файл и рсар файл в NS-3.....	17
2.2.2 Рсар файл в NS-3	18
3.Сравнение симуляторов NS-2 и NS-3 на примере одного сценария	18
3.1 Описание тестируемого протокола	19
3.2 Сравнительный анализ общего сценария	20
3.2.1 Описание алгоритма работы программ на NS-2.....	21
3.2.2 Описание алгоритма работы программы на NS-3	22
3.2.3 Сравнение метрик NS-2 и NS-3	23
4.VANET	25
4.1. Сравнительная характеристика симуляторов VANET.....	25
Заключение	27
Список источников	29
Листинг	30

Введение

Современные сетевые симуляторы широко используются как в исследовании поведения и производительности уже существующих протоколов, так и в создании новых. Такой подход даёт возможность исследователям сэкономить деньги и время, так как проводить эксперименты на реальном оборудовании может быть не только сложно по техническим причинам, но и дорого стоить. Поэтому симуляция остаётся наиболее распространённым подходом к разработке и тестированию протоколов. Однако сами симуляторы сетей построены на теории очередей и дискретных событиях, у них может отсутствовать некоторые детали в документации: выше сказанное влияет на качество тестируемого или создаваемого продукта.

На сегодняшний день существуют множество симуляторов сетей: NS-2, NS-3, OMNET++, OPNET, Glomosim, QUALNET, JIST/SWANS. Но только NS-2, NS-3 и OMNET++ из перечисленных бесплатные и с открытым исходным кодом. Мы отдадим предпочтение последним двух симуляторам по причине, описанной выше.

Цель работы

Необходимо провести сравнительный анализ эмуляторов ad-hoc сетей, в течение которого необходимо исследовать надёжность получаемых результатов.

Задачи

1. Кратко описать симуляторы, не вошедших в детальное рассмотрение
2. Сравнить архитектуры симуляторов NS-2 и NS-3
3. Провести сравнительный анализ общего сценария на NS-2 и NS-3
4. Описать VANET-симуляторы и показать различия между MANET

1. Хронология сетевых симуляторов

В этом разделе мы коротко опишем симуляторы, не рассмотренные на практике. Сами симуляторы представлены в хронологическом порядке.

SLAM II

Самый первый сетевой симулятор был создан в 1985, и он назывался SLAM II. В нём впервые применили дискретно-событийную модель.

OPNET

Позже в 1986 был представлен OPNET, написанный на C и C++. Он разрабатывался, в частности, для военных целей. Ниже представлены некоторые из возможностей этого симулятора:

- Доступен удобный инструмент тестирования протоколов и на готовых сценариях
- Графический интерфейс
- Встроенный анализатор топологии сети

Однако присутствуют и ограничения:

- Поддерживается малое количество протоколов
- Отсутствует хорошая беспроводная мобильность нод

OMNET++

Затем в 1993 был создан OMNET++. Это объектно-ориентированный дискретный-событийный симулятор, написанный на C++. Ниже представлены некоторые из возможностей этого симулятора:

- Поддержка масштабирования сетей
- Полная библиотека различных реализаций топологий
- Графические инструменты для анализа результата работы эмуляции
- Поддержка моделирования трафика телекоммуникационных сетей
- Содержит библиотеки с готовыми к использованию контейнерами и разного рода очередями

Однако присутствуют и ограничения:

- Неточный анализатор производительности

- Неполная документация
- Недостаточное количество поддерживаемых протоколов

Glomosim

В 1998 году был представлен Glomosim. Этот масштабируемый симулятор дискретных событий поддерживает эмуляцию как проводных, так и беспроводных сетей, а также использует модель OSI. При его разработки использовали C-подобный язык PERSEC, позволяющий использовать многопоточность. Этот симулятор имеет следующие возможности:

- В силу использования PERSEC, могут запускаться различные асинхронные протоколы
- Поддержка беспроводных протоколов
- Поддерживается симуляция очень больших топологий

Однако присутствуют и ограничения:

- Неполная документация
- Для многопоточной разработки потребуется установка PERSEC

QualNet

В 1999 году был создан QualNet компанией Scalable Network Technologies (SNT) для военных проектов, например, JTRS. Ниже перечислены некоторые отличительные особенности в положительном ключе:

- Пользовательский интерфейс написан на Java
- Обеспечивает сравнительную оценку производительности на каждом уровне OSI
- Поддержка анимации в топологии
- Поддержка очень большого количество узлов в топологии (до 10 тысяч)
- Хорошая документация

Однако присутствуют и ограничения:

- Очень дорогая разработка, необходима покупка лицензии
- Плохая оптимизация

- Очень сложный процесс установки на Linux

JIST/SWANS

В 2005 году был выпущен JIST/SWANS. JIST – это высокопроизводительный дискретный симулятор. Он написан на Java. SWANS – это масштабируемый симулятор беспроводной сети, построенный на платформе JIST. Ниже приведены некоторые особенности симулятора:

- Возможность симулировать большие топологии из-за использования Java

Однако присутствуют и ограничения:

- Разработка прекращена сразу после выпуска в 2005

В следующем пункте мы подробно разберём NS-2 и NS-3.

2. Архитектура симуляторов NS-2 и NS-3.

Сетевые симуляторы имеют разную направленность и используются в разных областях исследований. Следовательно, они различаются по своей архитектуре. Архитектура системы является центральным элементом, который позволяет создавать сложные системы моделирования. Поэтому важно рассматривать архитектуру как первый критерий сравнения различных сред моделирования. В этом разделе мы рассмотрим характеристики архитектур NS-2 и NS-3.

2.1 Архитектура NS-2

NS-2 – это сетевой симулятор, основанный на дискреционных событиях. Дискретно-событийность в данном случае означает отсутствие привязки ко времени. То есть симулятору достаточно соблюдать последовательность наступления событий, при этом не важно, какой временной промежуток будет между событиями. Симулятор NS-2 состоит из двух частей. Одна из них написана на языке C++ и должна быть перекомпилирована в случае внесения изменений или дополнений с помощью make, другая написана на

интерпретируемом языке OTcl (объектно-ориентированное расширение языка сценариев Tcl) и, соответственно, не требует компиляции. Планировщик событий и большинство компонентов сетей реализованы на C++, так как важна скорость выполнения.

Взаимодействие между частями, написанными на таких принципиально разных языках программирования, осуществляется согласно спецификации, определяющей способ обращения из tcl-скрипта к любому методу классов компилируемой иерархии и возвращение назад результатов, а также способ обращения из программы на C++ к любому методу, описанному в Tcl-скрипте.

Специфика связи между двумя языками может быть описана в двух случаях:

1. Передача параметра из C++ в Otcl

```
int MyObject::command(int argc,const char* const* argv)
{
    Tcl& tcl = Tcl::instance ();
    if (argc==2) {
        if(strcmp(argv[1],” get-delay”) == 0) {
            tcl.resultf(“%2.2f\n,delay_”);// переменная из C++
            delay_ теперь хранится в $d
            return (TCL_OK);
        }
    }
    return (TclObject::command(argc,argv);
}
```

Тогда как в .tcl принимается значение вот так:

```
set ns [new Simulator]
set obj [new MyOtclObject]
set d [$obj get-delay] // связана с tcl.resultf
puts “The variable d contains”
```

```
puts "$d"
```

2. Передача параметра из Otcl в C++

```
set ns [new Simulator]
```

```
set d 50
```

```
set obj [new MyOtclObject] // когда создаём MyOtclObject вызывается  
конструктор C++ класса MyObject
```

```
puts "The variable d contains"
```

```
puts "$d"
```

```
MyObject::MyObject()
```

```
{
```

```
    Tcl& tcl = Tcl::instance();
```

```
    Tcl.eval ("set d")
```

```
    delay_ = atoi(tcl.result()); // теперь переменная delay содержит 50
```

```
}
```

По замыслу создателей ns-2, все методы, имеющие дело с отдельными пакетами и потому требующие высокого быстродействия, относятся к компилируемой части. Интерпретируемая же часть отвечает за менее частые события, чем передача пакетов, обеспечивающие управление ходом моделирования, и манипуляцию объектами, описанными в компилируемой части. Также, Tcl-скриптом является собственно описание модели сети, подлежащей исследованию. Такой подход позволяет быстро построить требуемую модель сети с помощью скриптового языка OTcl без необходимости вникать в структуру компилируемой части ns-2. В случае, если необходима модификация или дополнение компилируемой части, это может быть сделано путем добавления (или изменения) C++ кода и перекомпиляции через make. Единственный недостаток такого подхода – трудности при изучении системы и отладке программ (моделей), возникающие вследствие использования двух языков.

2.1.1 Основные понятия в NS-2

В NS-2, сеть представляет собой совокупность сетевых объектов (Network Objects), каждый из которых способен определенным образом реагировать на некоторое множество событий (Events). Обо всех событиях сетевые объекты извещаются так называемым планировщиком (Scheduler), который содержит хронологическую таблицу событий. Каждое событие имеет обязательные атрибуты: время наступления и сетевой объект, к которому относится это событие. В системе NS-2 пакет – это событие, сгенерированное в объекте-получателе другим сетевым объектом (отправителем). Так как обмен пакетами – это единственный способ взаимодействия между сетевыми объектами, пакет – это единственный тип «внутреннего» события в системе. Все остальные типы событий генерируются пользователем при описании модели. Они называются at-событиями. Такие события предназначены для управления ходом моделирования (включение/выключение генератора трафика, обрыв/восстановление линии связи и т.п.).

2.1.2 Планировщик событий в NS-2

Планировщик по очереди (в хронологическом порядке) выбирает из таблицы события и извещает соответствующие сетевые объекты о наступлении этих событий, сообщая также момент времени, в который произошло это событие. С точки зрения программиста, такое «извещение о событии» представляет собой просто вызов метода-обработчика для соответствующего объекта. Вся сопутствующая информация передается через параметры этого метода. Помещать события в таблицу планировщика могут как сами сетевые объекты, так и пользователь при описании модели. Как говорилось выше, в первом случае событие называется пакетом, во втором – at-событием. Таким образом, любое взаимодействие между сетевыми объектами осуществляется через посредство планировщика. Отправка пакета одним сетевым объектом другому есть ни что иное, как постановка отправителем в таблицу события, адресованного получателю и помеченного тем моментом времени, когда пакет должен до этого получателя прийти

(возможно, с учетом задержки при передаче). Получение пакета, соответственно, представляет собой извещение планировщиком получателя о событии и обработка этого события получателем. При этом (если речь не идет о планировщике реального времени) модельное время не имеет никакого отношения к реальному. Планировщик извлекает события из таблицы, обращая внимание лишь на порядок их следования. Интервал между обработкой последовательных событий определяется лишь скоростью их обработки компьютером и не зависит от того, какими моментами модельного времени помечены эти события в таблице. Модельное время находит свое отражение лишь в трассировочных файлах, куда сетевые объекты записывают моменты отправки и получения пакетов (естественно, в модельном времени).

2.1.3 Абстракции транспортного и прикладного уровней: Агенты и Приложения

Для создания TCP-соединения необходимо создать два агента: TCP-передатчик и TCP-приемник. TCP-приемник лишь принимает пакеты и посылает назад уведомления о получении (ACK). Поэтому такой агент в системе один: TCP-Sink. Передатчик же может вести себя различным образом в зависимости от порядка и моментов получения им уведомлений от адресата о получении отправленных им пакетов. Соответственно, в системе ns-2 имеется несколько вариантов TCP-передатчика, моделирующих различные реализации протокола TCP (TCP, TCP/Tahoe, TCP/Reno, TCP/Vegas), использующие разные комбинации алгоритмов TCP (Slow Start, Congestion Avoidance, Fast Recovery, Fast Retransmit). С протоколом UDP все обстоит гораздо проще, потому что уведомления о получении там не предусмотрены. Для передачи используется агент UDP, а для приема – агент Null, просто принимающий и отбрасывающий все приходящие пакеты. За прикладной уровень в ns-2 отвечают приложения (Applications). Приложение прикрепляется к агенту и служит для создания трафика. В ns-2 имеются приложения, моделирующие трафик, характерный для реальных протоколов прикладного уровня (FTP и Telnet), а также абстрактные генераторы трафика

различного типа (например, CBR – простейший генератор трафика с постоянным темпом выдачи пакетов). Приложения запускаются и останавливаются пользовательскими at-событиями.

2.1.4 Описание trace файла в NS-2

Файл трассировки создаётся симулятором для хранения и ведения отчётности о прошедшей симуляции, например, производительности, доставки пакетов и или их потери. В итоге полученный файл можно распарсить с помощью awk-скрипта и получить необходимую информацию. Указание на создание такого файла происходит на языке Otcl:

```
# Open the trace file
set nf [open out.tr w]
$ns trace-all $nf
// $ns use-newtrace // необходимо для беспроводных в новом формате
```

Таким образом мы говорим открыть файл out на запись с расширением “.tr”, “nf” является токеном трассировочного файла. Так же мы указываем симулятору трассировать каждый пакет по каждому событию в топологии и для этого мы указываем токен трассировочного файла.

Для того чтобы закрыть файл воспользуемся:

```
# Define finish procedure
proc finish {} {
    global ns nf
    $ns flush-trace
    close nf
    exit 0
}
```

Командой flush мы окончательно запишем в файл результаты и закроем файл через токен.

2.1.5 Новый формат трассировочного файла

Для того чтобы сгенерировать файл в новом формате беспроводных сетей нужно использовать \$ns use-newtrace. Опишем новый формат:

```
г -t 0.037790333 -Hs 0 -Hd 0 -Ni 0 -Nx 0.00 -Ny 0.00 -Nz 0.00 -Ne 0.000424 -NL AGT -Nw --- -Ma 13a -Md 0 -Ms 9 -Mt 000 -Is 9.0 -Id 0.0 -It ack -Il 40 -If 0 -Ii 8 -Iv 30 -Pn tcp -Ps 3 -Pa 0 -Pf 1 -Po 0
с -t 0.037790333 -Hs 0 -Hd -2 -Ni 0 -Nx 0.00 -Ny 0.00 -Nz 0.00 -Ne 0.000424 -NL AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 9.0 -It tcp -Il 1064 -If 0 -Ii 14 -Iv 32 -Pn tcp -Ps 7 -Pa 0 -Pf 0 -Po 0
г -t 0.037790333 -Hs 0 -Hd -2 -Ni 0 -Nx 0.00 -Ny 0.00 -Nz 0.00 -Ne 0.000424 -NL RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 9.0 -It tcp -Il 1064 -If 0 -Ii 14 -Iv 32 -Pn tcp -Ps 7 -Pa 0 -Pf 0 -Po 0
с -t 0.037790333 -Hs 0 -Hd -2 -Ni 0 -Nx 0.00 -Ny 0.00 -Nz 0.00 -Ne 0.000424 -NL AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 9.0 -It tcp -Il 1064 -If 0 -Ii 15 -Iv 32 -Pn tcp -Ps 8 -Pa 0 -Pf 0 -Po 0
г -t 0.037790333 -Hs 0 -Hd -2 -Ni 0 -Nx 0.00 -Ny 0.00 -Nz 0.00 -Ne 0.000424 -NL RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 9.0 -It tcp -Il 1064 -If 0 -Ii 15 -Iv 32 -Pn tcp -Ps 8 -Pa 0 -Pf 0 -Po 0
с -t 0.037790333 -Hs 0 -Hd 9 -Ni 0 -Nx 0.00 -Ny 0.00 -Nz 0.00 -Ne 0.000424 -NL RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 9.0 -It tcp -Il 1064 -If 0 -Ii 14 -Iv 30 -Pn tcp -Ps 7 -Pa 0 -Pf 0 -Po 0
```

Рис.1, трассировочный файл в новом формате

Такой файл имеет следующее описание:

s – отправка пакета

г - принятие пакета

d – сброс пакета

f – пакет отправлен

с – коллизия пакета на МАС уровне

t – время, когда началась трассировка пакета

Hs – идентификатор узла-источника

Hd – идентификатор следующего узла-назначения

Ni – идентификатор узла

Nx,Ny,Nz – координаты, где расположен узел

Ne – энергия узла

Nl – трассировочный уровень

Nw – причина события

AGT - агент

Is - Адрес источника порта источника

Id - Адрес назначения порта назначения

П – размер пакета

If – номер потока

Ii – уникальный идентификатор

Iv – время жизни (TTL) на следующий узел

Ma – длительность пребывания пакета на уровне МАС

Md - целевой Ethernet адрес

Ms - Ethernet источника

Mt – тип Ethernet

PACKET INFORMATION

Pm - MAC адрес источника

Ps – адрес источника

Pa - MAC адрес узла назначения

Po – оптимальное количество на отправку

-Ps – номер последовательности

Pf – провал в отправке пакета(потеря)

2.1.6 Старый формат трассировочного файла

Однако если не использовать `$ns use-newtrace` в скрипте, то мы будем иметь трассировочный файл в старом формате:

```
s 21.500275000 _0_ MAC --- 0 AODV 106 [0 ffffffff 0 800] ----- [0:255 -1:255 30 0] [0x2 1 4 [1 0] [0 10]] (REQUEST)
```

Рис.2, старый формат в беспроводных сетях

- Первое значение `r` или `s` – это отправка или принятие в определенное время (следующее значение)
- Уровень MAC
- Уникальный идентификатор `0`, полезная нагрузка – тип протокола, размер пакета 106 байт.
- MAC адрес источника и назначения `0` и `ffffff` соответственно
- Пакет находится в сети Ethernet, то есть 800
- IP-адрес источника и назначения равны `0` и `1` соответственно
- Порты источника и назначения 255
- TTL и номер следующего узла перехода составляют 30 прыжков и `0` соответственно
- `0x2` – это пометка протокола AODV
- Количество отсчетов скачков равно `1`, а идентификатор широковещательной передачи равен `4`

- Номер узла назначения и порядковый номер равны 1 и 0 соответственно
- Строка «(REQUEST)» подтверждает, что это пакет RREQ

2.2 Архитектура NS-3

Аналогично NS-2, NS-3 – это сетевой симулятор, основанный на дискреционных событиях. Однако разработчики отказались от обратной совместимости с NS-2, поэтому они создали этот симулятор с нуля на C++.

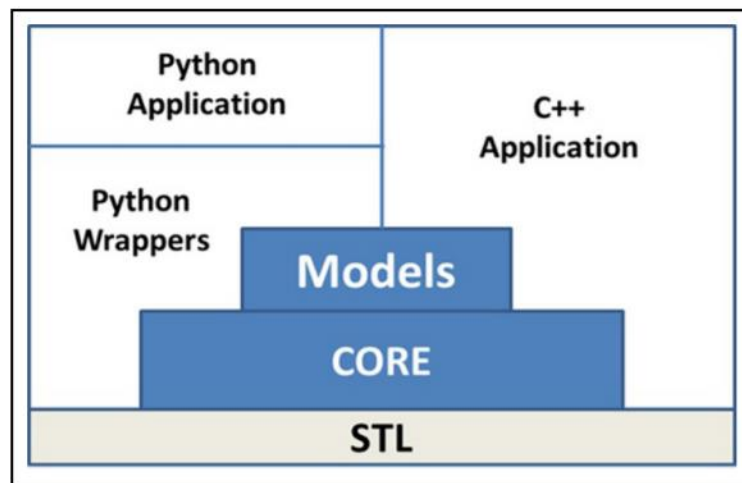


Рис.3, модульная архитектура NS-3

NS-3 имеет модульную архитектуру, содержащую:

- Ядро, в состав которого входят отладчик, генератор случайных чисел
- Симулятор, включающий в себя события, планировщик
- “Common” библиотека управляет созданием трассировок
- Библиотека узлов определяет классы объектов протоколов
- Протоколы, затрагивающие разные уровни OSI

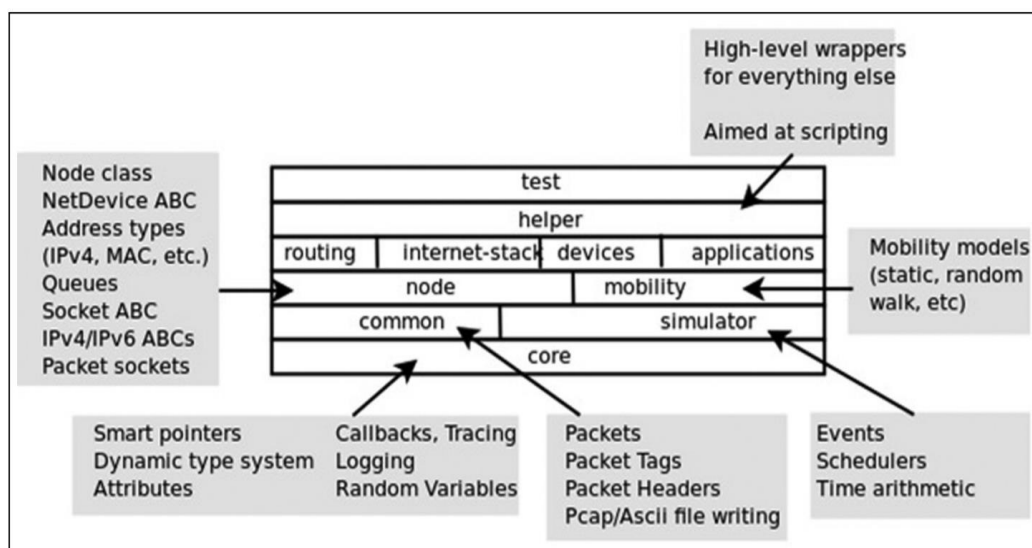


Рис.4, модульность в C++ приложениях

Беспроводная сетевая модель в NS-3

Архитектура модели Wi-Fi интерфейса узла является модульной и подразделяется на три уровня:

- PHY layer model (модель физического уровня) обеспечивает прием и передачу кадров из канала и в канал соответственно, обработку ошибок.
- MAC low models. Модели нижнего MAC уровня контролируют доступ к среде передачи данных, выполняют обработку фрагментированных кадров и обрабатывает очереди пакетов при необходимости.
- MAC high models. Объединяет модели, в задачи которых входит генерация маяков (Beaconing) и зондирование (Probing) при работе в режиме BSS, выполнение ассоциации узлов и точек доступа и соответственно сама реализация режима работы устройства:
- Узлы IBSS. Класс – AdhocWifiMac (именно этот режим нас интересует);
- Класс “ns.wifi.YansWifiChannelHelper”.С помощью этого класса мы описываем модели затухания сигнала и задержки

Схема модуля с описанными выше уровнями и соответствующими классами NS-3 представлена на Рисунке 5:

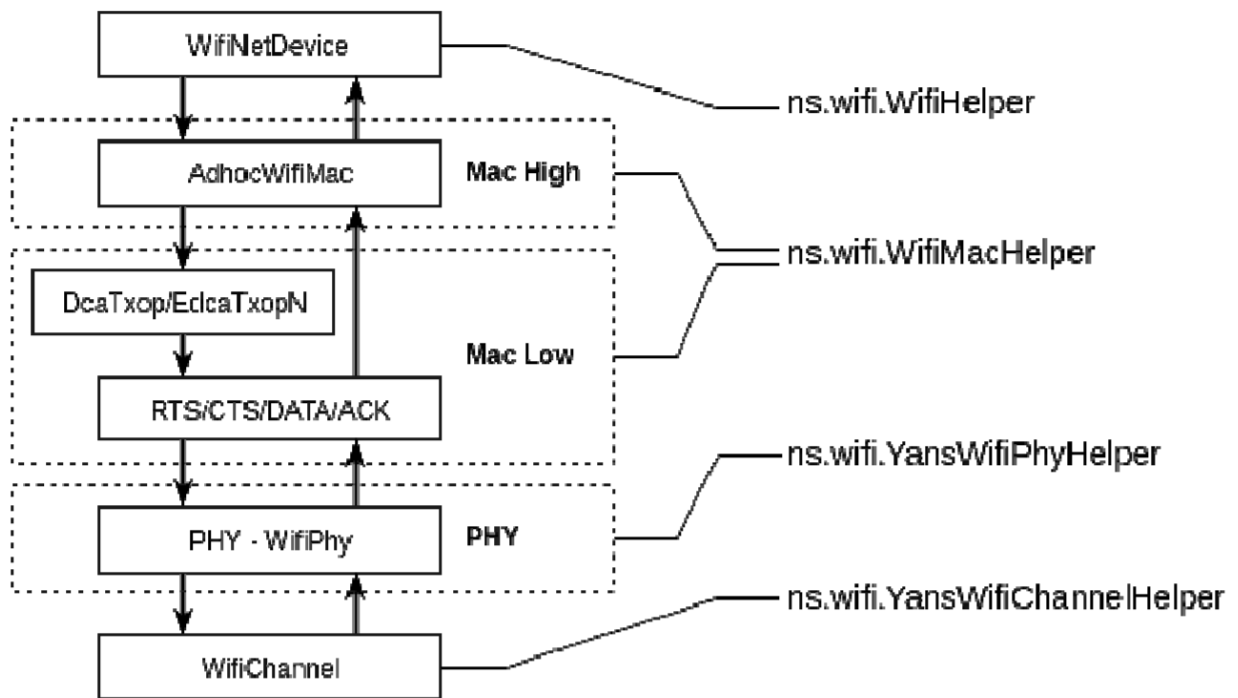


Рис.5, иерархия моделирования сети

Подготовка топологии к практическому сценарию.

Теперь описанные конструкции сетевых настроек, мы дополняем для создания сценария AODV:

1. PHY – Используем класс “ns.wifi.YansWifiPhyHelper”. На этом этапе мы конфигурируем модель обработки ошибок на основе сведений получаемых из среды передачи данных. В этом моменте связываются физический уровень устройства с каналом передачи данных. По умолчанию применяется модель обработки ошибок – “ns3:: NistErrorRateModel”.
2. Для MAC Low и MAC High используется класс “ns.wifi.WifiMacHelper”, который помогает конфигурировать необходимые параметры, в том числе задать режим работы модуля Ad-hoc.
3. Используя класс “ns.wifi.WifiHelper” мы настраиваем режим, в котором будет работать Wi-Fi модуль. В NS-3 предлагаются стандарты 802.11a/b/n/ac, n – в диапазоне 2.4 и 5 GHZ. При проведении эксперимента для определенности будем использовать 802.11n_2.4GHZ.

На данном этапе модель сети представляет собой набор узлов с установленными сетевыми модулями. Для ее работоспособности необходимо добавить к узлам набор сетевых протоколов. Для добавления на узлы Ipv4 набора протоколов и в том числе протокола маршрутизации, используется класс “InternetStackHelper”. Именно в этом моменте будут отличаться модели сети при проведении эксперимента с протоколом AODV.

2.2.1 Трассировочный файл и рсар файл в NS-3

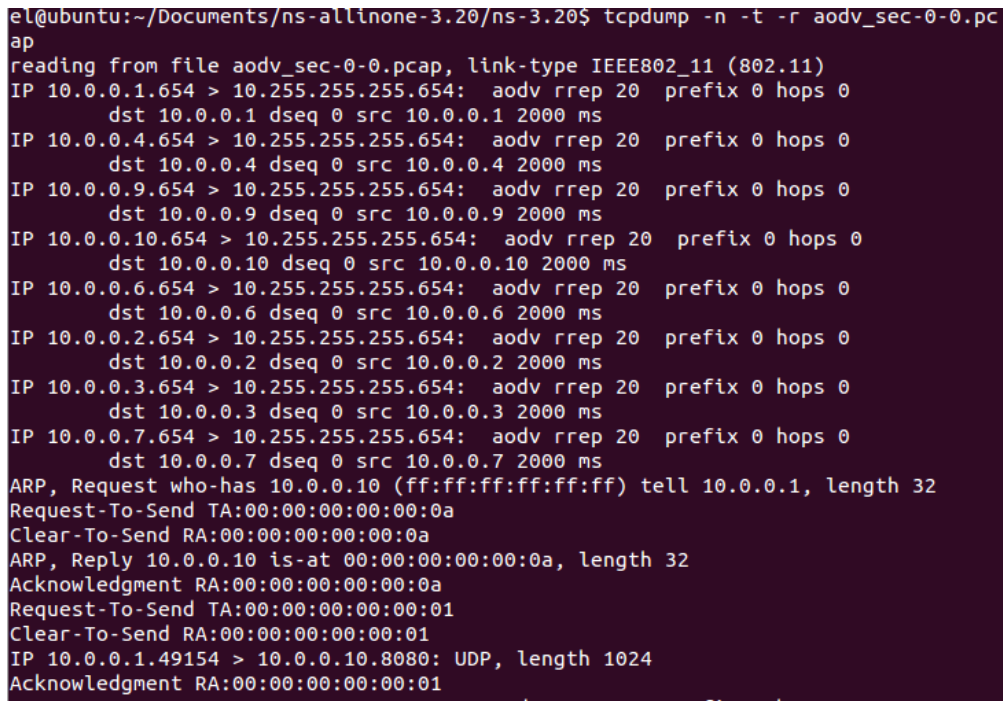
Трассировочный файл генерируется после установки на физический уровень, где установлен YansWifiHelper, EnableAsciiAll. После этого сгенерируется .tr файл в формате, который представлен ниже:

```
r      8.02714      /NodeList/9/DeviceList/0/$ns3::WifiNetDevice/Phy/State/RxOk
ns3::WifiMacHeader (DATA ToDS=0, FromDS=0, MoreFrag=0, Retry=0,
MoreData=0      Duration/ID=0usDA=ff:ff:ff:ff:ff:ff,      SA=00:00:00:00:00:01,
BSSID=00:00:00:00:00:01, FragNumber=0, SeqNumber=17) ns3::LlcSnapHeader
(type 0x800) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 1 id 16
protocol 17 offset (bytes) 0 flags [none] length: 48 10.0.0.1 > 10.255.255.255)
ns3::UdpHeader (length: 28 654 > 654) ns3::aodv::TypeHeader (RREP)
ns3::aodv::RrepHeader (destination: ipv4 10.0.0.1 sequence number 0 source ipv4
10.0.0.1 lifetime 2000 acknowledgment required flag 0) ns3::WifiMacTrailer ()
```

Здесь был совершен приём пакета в восьмую секунду эмуляции по беспроводной сети по протоколу IPv4 с указанными ip-адресами

2.2.2 Рсар файл в NS-3

Так же можно создать и рсар файл с помощью EnableRсар и выбрать нужные узлы, затем рассмотреть его с помощью tcpdump:



```
el@ubuntu:~/Documents/ns-allinone-3.20/ns-3.20$ tcpdump -n -t -r aodv_sec-0-0.pcap
ap
reading from file aodv_sec-0-0.pcap, link-type IEEE802_11 (802.11)
IP 10.0.0.1.654 > 10.255.255.255.654: aodv rrep 20 prefix 0 hops 0
dst 10.0.0.1 dseq 0 src 10.0.0.1 2000 ms
IP 10.0.0.4.654 > 10.255.255.255.654: aodv rrep 20 prefix 0 hops 0
dst 10.0.0.4 dseq 0 src 10.0.0.4 2000 ms
IP 10.0.0.9.654 > 10.255.255.255.654: aodv rrep 20 prefix 0 hops 0
dst 10.0.0.9 dseq 0 src 10.0.0.9 2000 ms
IP 10.0.0.10.654 > 10.255.255.255.654: aodv rrep 20 prefix 0 hops 0
dst 10.0.0.10 dseq 0 src 10.0.0.10 2000 ms
IP 10.0.0.6.654 > 10.255.255.255.654: aodv rrep 20 prefix 0 hops 0
dst 10.0.0.6 dseq 0 src 10.0.0.6 2000 ms
IP 10.0.0.2.654 > 10.255.255.255.654: aodv rrep 20 prefix 0 hops 0
dst 10.0.0.2 dseq 0 src 10.0.0.2 2000 ms
IP 10.0.0.3.654 > 10.255.255.255.654: aodv rrep 20 prefix 0 hops 0
dst 10.0.0.3 dseq 0 src 10.0.0.3 2000 ms
IP 10.0.0.7.654 > 10.255.255.255.654: aodv rrep 20 prefix 0 hops 0
dst 10.0.0.7 dseq 0 src 10.0.0.7 2000 ms
ARP, Request who-has 10.0.0.10 (ff:ff:ff:ff:ff:ff) tell 10.0.0.1, length 32
Request-To-Send TA:00:00:00:00:00:0a
Clear-To-Send RA:00:00:00:00:00:0a
ARP, Reply 10.0.0.10 is-at 00:00:00:00:00:0a, length 32
Acknowledgment RA:00:00:00:00:00:0a
Request-To-Send TA:00:00:00:00:00:01
Clear-To-Send RA:00:00:00:00:00:01
IP 10.0.0.1.49154 > 10.0.0.10.8080: UDP, length 1024
Acknowledgment RA:00:00:00:00:00:01
```

Рис.6 tcpdump читает рсар-файл

3. Сравнение симуляторов NS-2 и NS-3 на примере одного сценария

Дадим определение MANET – это мобильные беспроводные ad-hoc сети, которые не нуждаются в фиксированной топологии, так как их протоколы создают в разный момент времени необходимую топологию.

Прежде чем проводить сравнение двух симуляторов, нам необходимо определить один общий сценарий:

- 9 узлов, движущихся с одной общей скоростью
- Одно и тоже количество пакетов
- Одно и тоже направление движения узлов
- Генератор пакетов одинаковой мощности

3.1 Описание тестируемого протокола

AODV (англ. Ad hoc On-Demand Distance Vector) — протокол динамической маршрутизации для мобильных ad-hoc сетей. Этот протокол устанавливает маршрут до адресата по требованию, то есть является реактивным протоколом маршрутизации. Далее будет описана работа данного протокола. Допустим некоторому узлу, пусть он будет называться инициатором, понадобилось передать данные другому узлу, пусть он будет узлом назначения, но маршрут между ними не составлен, тогда узел-инициатор отправляет широковещательный запрос RREQ (Route Request), чтобы данный маршрут отыскать. Данный запрос отправляется всем соседям рассматриваемого узла, который пересылается по цепочке далее – от соседа к соседу. Когда запрос, имеющий в своем заголовке данные об узле инициаторе, проходит через сеть, все узлы, получившие данный кадр, строят до этого инициатора свой маршрут. Таким образом, запрос продвигается по сети до узла назначения, а все узлы, которые он посетил, составляют до него маршрут. Узел назначения в итоге получает запрос RREQ (запрос создания маршрута) и отправляет юникастовый ответ RREP (Route Replay) по маршруту, который выстроился во время распространения запроса. По итогам такого обмена выстраивается двунаправленный маршрут между узлами. В ответ на запрос промежуточным узлом, отправляется так называемый «беспричинный» ответ не инициатору, а узлу назначения. Такой обмен данными нужен для того, чтобы построить маршрут обратно – от узла назначения до узла инициатора. Во время такого интенсивного обмена информацией в сети, в узлах могут оказаться маршруты разной степени давности, в таком случае, очевидно, что для построения маршрутов целесообразней использовать более свежую информацию. Но как определить узлу эту самую свежесть маршрута? Для этого в данном протоколе были введены порядковые номера узлов. Один из авторов протокола AODV, взял идею о нумерации из другого своего детища, протокола DSDV. Суть идеи заключается в следующем: каждому узлу

присваивается собственный порядковый номер, значением которого он может управлять самостоятельно, этот номер также появляется во всей информации о маршрутах, где содержится данный узел, и даже хранится в таблицах маршрутизации на узлах маршрутизаторах и узлах координаторах.

3.2 Сравнительный анализ общего сценария

В этом сравнении мы покажем, как разными симуляторами эмулируется пропускная способность и доставка пакетов на одном и том же сценарии. Мы использовали аниматор NAM, чтобы сгенерировать топологию в NS-2:

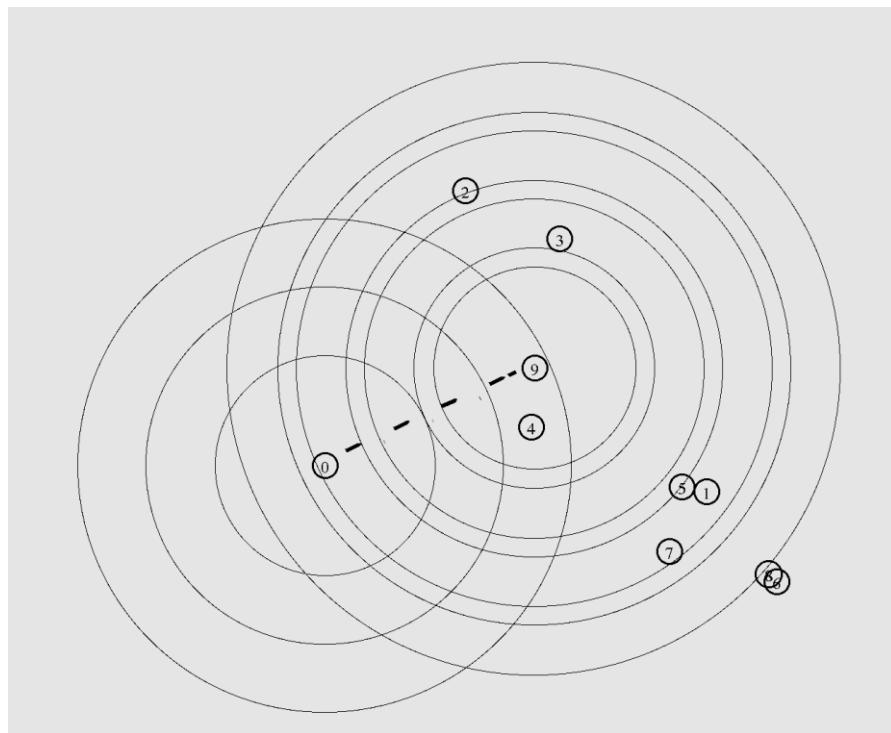


Рис.7, визуализация AODV в NS-2

Тогда как NS-3 поставляется с аниматором NetAnim на Python, и топология выглядит следующим образом:

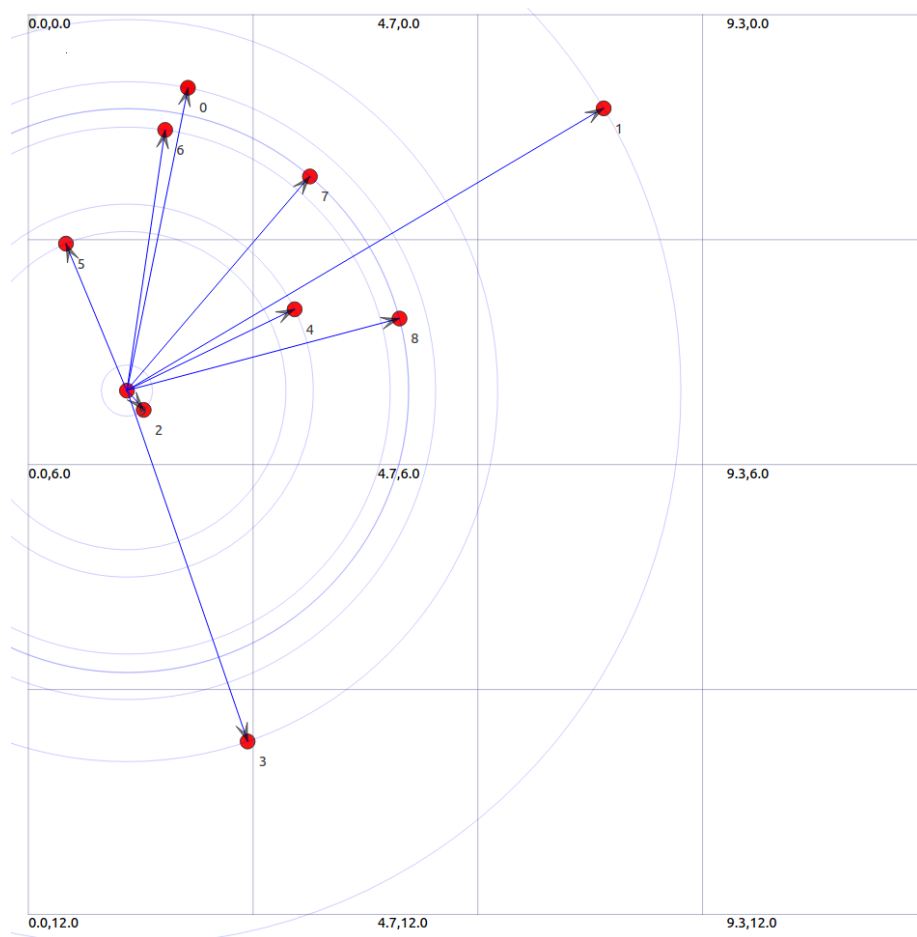


Рис.8, визуализация AODV в NS-3

3.2.1 Описание алгоритма работы программ на NS-2

OTCL

Созданный скрипт на OTCL в самом начале должен иметь настройки для симуляции, эти настройки будут вызваны в нужный момент как функция. Мы определили следующие функции для симуляции:

- Тип канала – беспроводной
- Тип MAC – IEEE 802.11
- Тип работы очереди – DropTail(сброс пакетов, если максимум приёма превышен)
- Тип протокола – AODV
- Размер поля анимации – 1000x1000
- Время симуляции – 10 с.

Затем загружаем поле и открываем трассировочный и анимационный файлы на запись в новом формате. Задаём параметры узлов путём вызова вышеуказанных функций. После расставляем изначальное положение узлов на

карте. Затем генерируем движение узлов по карте с помощью at-событий в указанное время на указанные позиции. Для генерации TCP-трафика создадим исток и сток (Агентов) и соединим их, указав размер передаваемых пакетов. Потом указываем время начала и конца симуляции. Чтобы завершить программу, нужно закрыть все файлы: трассировочный и анимационный, и выйти в указанное в stop время.

C++

Код на C++ для протокола AODV уже сгенерирован в NS-2, нам осталось только добавить счётчик пакетов, узел-отправителя пакета и узел-назначение. Для этого мы в функции forward добавляем нужный printf.

```
num_nodes is set 10
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5, distCST_ = 550.0
SORTING LISTS ...DONE!
I am packet no 0 and from node number 1
I am packet no 0 and from node number 2
I am packet no 0 and from node number 3
I am packet no 0 and from node number 4
I am packet no 0 and from node number 5
I am packet no 0 and from node number 6
I am packet no 0 and from node number 7
I am packet no 0 and from node number 8
I am packet no 0 and from node number 0
I am packet no 0 and from node number 9
I am packet no 1 and from node number 0
I am packet no 2 and from node number 0
I am packet no 1 and from node number 9
I am packet no 3 and from node number 0
I am packet no 4 and from node number 0
I am packet no 2 and from node number 9
I am packet no 3 and from node number 9
I am packet no 5 and from node number 0
I am packet no 6 and from node number 0
I am packet no 4 and from node number 9
I am packet no 5 and from node number 9
I am packet no 6 and from node number 9
I am packet no 7 and from node number 0
I am packet no 8 and from node number 0
el@ubuntu:~$
```

Рис.9, вывод происходящих событий

3.2.2 Описание алгоритма работы программы на NS-3

Алгоритмы на NS-3 можно писать либо на Python, либо на C++. Выбор пал на C++. Сначала укажем интервал послыки пакетов, их количество и общее время выполнения. Затем создадим узлы с помощью NodeContainer. Затем установим размер и форму поля в виде прямоугольника, чтобы позволить узлам случайно перемещаться по карте. Для добавления на узлы Ipv4 набора протоколов и в том числе протокола маршрутизации, используется класс InternetStackHelper. Затем включаем протокол маршрутизации с помощью SetRoutingHelper. После установим исток и сток на 0 и 9 узел соответственно и запустим трафик, и планировщик.

```

Creating 10 nodes 100 m apart.
Packet sent - 1
Received packet - 1 and Size is 1024 Bytes.
Packet sent - 2
Received packet - 2 and Size is 1024 Bytes.
Packet sent - 3
Received packet - 3 and Size is 1024 Bytes.
Packet sent - 4
Received packet - 4 and Size is 1024 Bytes.
Packet sent - 5
Received packet - 5 and Size is 1024 Bytes.
Packet sent - 6
Received packet - 6 and Size is 1024 Bytes.
Packet sent - 7
Received packet - 7 and Size is 1024 Bytes.
Packet sent - 8
Received packet - 8 and Size is 1024 Bytes.
Packet sent - 9
Received packet - 9 and Size is 1024 Bytes.
-----Results output ID wise -----

```

Рис.10, отправка пакетов в NS-3

3.2.3 Сравнение метрик NS-2 и NS-3

Для определения средней пропускной способности и коэффициент доставки пакетов на канале между 0 и 9 узлами, воспользуемся awk-скриптом, доступном на официальном сайте разработчиков NS, который распарсит трассировочный файл и выдаст результат:

```

The throughput in kbps is 2.328638
el@ubuntu:~$

```

Рис.11, пропускная способность в NS-2

```

Packet Delivery Ratio is 0.687500
el@ubuntu:~$

```

Рис.12, коэффициент доставки пакетов в NS-2

В NS-3, модуль FlowMonitor обеспечивает сбор данных, о производительности сети позволяя в автоматическом режиме определять все потоки трафика, проходящие через сеть, осуществлять выборку набора данных необходимых для последующего проведения оценки производительности сети в контексте поставленной задачи относительно каждого потока. Поэтому воспользуемся этим для определения метрик. Для этого установим итератор на начало ассоциативного массива и в цикле выведем номер потока, источник и получателя, отправленные пакеты и пропускную способность. Чтобы сосчитать последнее, мы должны перевести полученные получателем из байт

в биты путём умножения на 8, затем разделим на длительность передачи пакета, а потом всё разделим на 1024 и получим размер в кбит/с.

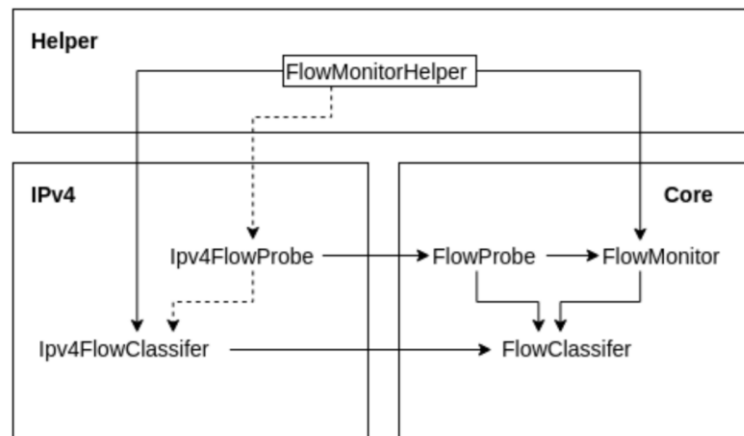


Рис.13, архитектура модуля FlowMonitor

```

-----Flow Id: 1
Src Addr 10.0.0.1 DST Addresss 10.0.0.10
Sent Packets = 9
Received Packets = 9
Delay = +16952168.0ns
Throughput = 9.24424 Kbps
-----Total Result of the simulation-----

Total Sent Packets = 9
Total Received Packets = 9
Average Throughput = 9.24424
Delay = +16952168.0ns
Total Flow Id = 1
Packet delivery ratio = 100 %
el@ubuntu:~/Documents/ns-allinone-3.20/ns-3.20$
  
```

Рис.14, коэффициент доставки пакетов и пропускная способность в NS-3

Разницу в полученных данных можно объяснить разной архитектурой обоих приложений.

4.VANET

Автомобильные самоорганизующиеся сети (VANETs) как и мобильные самоорганизующиеся сети (MANETs) возникают путем спонтанного создания беспроводной сети для обмена данными между транспортными средствами. Однако VANET симуляторы кардинально отличаются от MANET, потому что становится необходимым моделировать и учитывать топологии дорог, препятствия на обочине или дороге, модели транспортных потоков, изменение скорости и мобильности транспортных средств, светофоры, пробки на дорогах, поведение водителей. В дополнение к вышесказанному, в MANET узлы могут перемещаться случайно, относительно правил протокола, однако транспортные средства могут перемещаться только вдоль улиц, поэтому необходима модель дороги. Если мы создаем модель дороги, то нужно учитывать и эмулировать и ПДД. Вышеизложенное требует создание отдельных симуляторов для VANET.

4.1. Сравнительная характеристика симуляторов VANET

Прежде чем перейти к описанию VANET-симуляторов, необходимо определить генераторы мобильности.

VANET генераторы мобильности

VANET генераторы мобильности используются для повышения уровня реализма в VANET симуляторах. Они генерируют модели дорог, параметры сценариев, например, максимально допустимую скорость на сгенерированных дорогах, а также моделируют трафик.

Моделирование трафика

Моделирование трафика подразделяется на 3 категории:

- Макроскопические
- Мезоскопические
- Микроскопические

Макроскопические модели создаются в, например, METACOR. Это ПО позволяет моделировать трафик на топологиях большого масштаба. Она относится к трафику как к жидкости, то есть применяется

гидродинамические законы к моделируемому поведению машин. По этой причине такие модели предъявляют значительно меньшие требования к компьютеру, на котором проводится симуляция. Однако пропадает возможность анализа улучшения в транспортировке, как это возможно в микроскопической модели.

Мезоскопические модели создаются, например, в CONTRAM. CONTRAM комбинирует в себе свойства микроскопических и макроскопических имитационных моделей. Как и в микроскопических моделях, единицей потока трафика мезоскопических моделей является отдельный автомобиль. Однако их движения следуют подходу макроскопической модели, то есть скорость на дороге усредняется.

Микроскопическое моделирование трафика, например, в SUMO позволяет описать дорожное движение с аварийными ситуациями на дороге и избегать этого (модель Стефана Краусса). Позволяет детально моделировать движение одной машины, изменяющей полосу движения и меняющей тип дороги.

	TraNS	GrooveNet	NCTUns	MobiReal
Генератор Мобильности	SUMO	Встроенный, гибридный	Гибридный	Гибридный
Модели скорости	Установка для улицы	На основе нагрузки дороги	Случайная	Установка для улицы
Тип симуляции	Дискретная симуляция			
Модель Потока Трафика	Модель Стеффана Краусса	Обычное движение	Обычное движение	Обычное движение
Светофоры	Установка вручную	Установка вручную	Автоматически	Установка вручную

			на перекрёстках	
Модель перекрёстков	Согласно ПДД	Управляется только светофором	Управляется 4 светофорами	ПДД и светофоры
Модель поездки авто	Скрипт Случайное движение	Алгоритм Дейкстры	Скрипт	Скрипт
Встроенные Приложения	Определение опасности на дороге, динамическое Изменение маршрута	Определение опасности на дороге, динамическое Изменение маршрута	Нет	Нет
Протоколы	V2V, V2I	V2V, V2I	V2V, V2I	V2V, V2I

Таблица 1, сравнение VANET симуляторов

Заключение

В данном сравнении мы познакомились с теоретической и практической основой построения дискретных симуляторов сетей. В теоретической части мы детально изучили разницу в архитектурах обоих симуляторов, и убедились, что таким образом лишаемся обратной совместимости между ними. В практической части мы достигли поставленной цели, путём того, что выполнили задачи:

1. Симуляторы, не вошедшие в детальный разбор, оказались либо платными, либо с истёкшей поддержкой ПО и корректно не запускались на Windows 10. Поэтому они расценены, как неактуальные.

2. Теоретическое сравнение архитектур симуляторов показало, что отказ от агентов трафика и двух языков, позволили в NS-3 смоделировать беспроводные и проводные протоколы точнее, чем во второй версии.

3. Сравнение симуляторов NS-2 и NS-3 показало, что добавленные уровни OSI и измененная архитектура NS-3 лучше справляется с эмуляцией реального оборудования, поэтому каналы эмулируются точнее.

4. VANET требует отдельной эмуляции от MANET в силу генераторов трафика и загрузки карт.

Таким образом, мы изучили способы построения и эмуляции как мобильных, так и автомобильных ad-hoc сетей.

Список источников

1. Comparison of utility functions for routing in cognitive wireless ad-hoc networks–URL: <https://ieeexplore.ieee.org/document/5970478>
2. Performance Analysis of VANET Routing Protocols and Implementation of a VANETs– URL: <https://ieeexplore.ieee.org/document/8789319>
3. Design Doc. – URL: <https://www.nsnam.org/doxygen/classns3.html#details>
4. Introduction to Mobile Ad Hoc Networks – URL: <https://ieeexplore.ieee.org/document/7777323>
5. Comparison of utility functions for routing in cognitive wireless ad-hoc networks- URL: <https://ieeexplore.ieee.org/document/5970478>
6. Протоколы маршрутизации AODV и SAODV -URL: <https://lektsii.org/11-50758.html>
7. Vehicular Ad Hoc Networks – URL: <https://lektsii.org/11-50758.html>

Листинг

```

Phy/WirelessPhy set bandwidth_ 11Mb      ;#Data Rate
Mac/802_11 set dataRate_ 6Mb            ;#Rate for Data Frames
set val(chan) Channel/WirelessChannel    ;# channel type
set val(prop) Propagation/TwoRayGround   ;# radio-propagation model
set val(netif) Phy/WirelessPhy          ;# network interface type
set val(mac) Mac/802_11                  ;# MAC type
set val(ifq) Queue/DropTail/PriQueue     ;# interface queue type
set val(ll) LL                           ;# link layer type
set val(ant) Antenna/OmniAntenna         ;# antenna model
set val(ifqlen) 50                       ;# max packet in ifq
set val(nn) 10                           ;# number of mobilenodes
set val(rp) AODV                         ;# routing protocol
set val(x) 10000                         ;# X dimension of topography
set val(y) 10000                         ;# Y dimension of topography
set val(stop) 10.0                      ;# time of simulation end

#=====
# Initialization
#=====
#Create a ns simulator
set ns [new Simulator]

#Setup topography object
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)

#Open the NS trace file
set tracefile [open godhelp.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open godhelp.nam w]
$ns namtrace-all $namfile
$ns namtrace-all-wireless $namfile $val(x) $val(y)
set chan [new $val(chan)];#Create wireless channel
$ns use-newtrace

#=====
# Mobile node parameter setup
#=====
$ns node-config -adhocRouting $val(rp) \

```

```

-llType      $val(ll) \
-macType     $val(mac) \
-ifqType     $val(ifq) \
-ifqLen      $val(ifqlen) \
-antType     $val(ant) \
-propType    $val(prop) \
-phyType     $val(netif) \
-channel     $chan \
-topoInstance $topo \
    -energyModel "EnergyModel" \
    -initialEnergy 0.0331 \
    -txPower 0.9 \
    -rxPower 0.9 \
    -idlePower 0.6 \
    -sleepPower 1.0 \
-agentTrace  ON \
-routerTrace ON \
-macTrace    ON \
-movementTrace ON

```

```

#=====
#   Nodes Definition
#=====
#Create 10 nodes
set n0 [$ns node]
$n0 set X_ 0.0
$n0 set Y_ 0.0
$n0 set Z_ 0.0
$ns initial_node_pos $n0 20
set n1 [$ns node]
$n1 set X_ 0.0
$n1 set Y_ 0.0
$n1 set Z_ 0.0
$ns initial_node_pos $n1 20
set n2 [$ns node]
$n2 set X_ 0.0
$n2 set Y_ 0.0
$n2 set Z_ 0.0
$ns initial_node_pos $n2 20
set n3 [$ns node]
$n3 set X_ 0.0
$n3 set Y_ 0.0
$n3 set Z_ 0.0

```

```

$ns initial_node_pos $n3 20
set n4 [$ns node]
$n4 set X_ 0.0
$n4 set Y_ 0.0
$n4 set Z_ 0.0
$ns initial_node_pos $n4 20
set n5 [$ns node]
$n5 set X_ 0.0
$n5 set Y_ 0.0
$n5 set Z_ 0.0
$ns initial_node_pos $n5 20
set n6 [$ns node]
$n6 set X_ 0.0
$n6 set Y_ 0.0
$n6 set Z_ 0.0
$ns initial_node_pos $n6 20
set n7 [$ns node]
$n7 set X_ 0.0
$n7 set Y_ 0.0
$n7 set Z_ 0.0
$ns initial_node_pos $n7 20
set n8 [$ns node]
$n8 set X_ 0.0
$n8 set Y_ 0.0
$n8 set Z_ 0.0
$ns initial_node_pos $n8 20
set n9 [$ns node]
$n9 set X_ 0.0
$n9 set Y_ 0.0
$n9 set Z_ 0.0
$ns initial_node_pos $n9 20

#=====
#      Generate movement
#=====
$ns at 0.302916 "$n0 setdest 531.924000 84.678900 100 "
$ns at 0.580994 "$n0 setdest 374.14000 0.5 100 "
$ns at 0.625751 "$n0 setdest 348.745000 13.628900 100 "
$ns at 0.754892 "$n0 setdest 0.5 33.698500 100 "
$ns at 0.946701 "$n0 setdest 517.979000 63.507400 100 "
$ns at 1.26189 "$n0 setdest 248.409000 159.80500 100 "
$ns at 1.34352 "$n0 setdest 0.5 158.07200 100 "
$ns at 1.5 "$n0 setdest 476.158000 154.7500 100 "

```



```

$ns at 1.58972 "$n0 setdest 749.165000 152.84500 100 "
$ns at 1.75 "$n0 setdest 1133.67000 125.9500 100 "
$ns at 1.9313 "$n0 setdest 1568.6000 95.527800 100 "
$ns at 2.19395 "$n0 setdest 965.907000 15.730400 100 "
$ns at 2.53872 "$n0 setdest 484.273000 103.36800 100 "
$ns at 2.75 "$n0 setdest 41.5062000 69.867600 100 "
$ns at 2.76981 "$n0 setdest 0.5 66.727100 100 "

```

C++ код:

```

#include "ns3/aodv-module.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <cmath>

using namespace ns3;

int packetsSent = 0;
int packetsReceived = 0;
uint32_t SentPackets=0; //var for sending packets
uint32_t ReceivedPackets=0; // to receive packets
void ReceivePacket (Ptr<Socket> socket)
{
    Ptr<Packet> packet;
    while ((packet = socket->Recv ()))
    {
        packetsReceived++;
        std::cout<<"Received packet - "<<packetsReceived<<" and Size is "
                <<packet->GetSize ()<<" Bytes."<<std::endl;
    }
}

static void GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,

```

```

        uint32_t pktCount, Time pktInterval )
    {
        if (pktCount > 0)
        {
            socket->Send (Create<Packet> (pktSize));
            packetsSent++;
            std::cout<<"Packet sent - "<<packetsSent<<std::endl;

            Simulator::Schedule (pktInterval, &GenerateTraffic,
                                socket, pktSize,pktCount-1, pktInterval);
        }
        else
        {
            socket->Close ();
        }
    }
}

int main(int argc, char **argv)
{
    uint32_t size=10;
    double step=100;
    double totalTime=10/*2250*//*2175*/;
    bool asciitracing=true;
    int packetSize = 1024;
    int totalPackets = totalTime-1;
    double interval = 1;

    Time interPacketInterval = Seconds (interval);

    NodeContainer nodes;
    NetDeviceContainer devices;
    Ipv4InterfaceContainer interfaces;

    std::cout << "Creating " << (unsigned)size << " nodes " << step << " m apart.\n";
    nodes.Create (size);

    MobilityHelper mobility;
    mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
                              "Bounds", RectangleValue (Rectangle (-0.0000009, 100.0, -0.0000009,
100.0)));
    /*
    mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                              "MinX", DoubleValue (0.0),

```

```

        "MinY", DoubleValue (0.0),
        "DeltaX", DoubleValue (step),
        "DeltaY", DoubleValue (step),
        "GridWidth", UIntegerValue (size),
        "LayoutType", StringValue ("RowFirst")});
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
*/
mobility.Install (nodes);

NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
wifiMac.SetType ("ns3::AdhocWifiMac");
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
wifiPhy.SetChannel (wifiChannel.Create ());
WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                             "DataMode", StringValue ("OfdmRate6Mbps"), "RtsCtsThreshold",
                             UIntegerValue (0));
devices = wifi.Install (wifiPhy, wifiMac, nodes);

AodvHelper aodv;
InternetStackHelper stack;
stack.SetRoutingHelper (aodv);
stack.Install (nodes);
Ipv4AddressHelper address;
address.SetBase ("10.0.0.0", "255.0.0.0");
interfaces = address.Assign (devices);

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
Ptr<Socket> recvSink = Socket::CreateSocket (nodes.Get (size-1), tid);
InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 8080);
recvSink->Bind (local);
recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));

Ptr<Socket> source = Socket::CreateSocket (nodes.Get (0), tid);
InetSocketAddress remote = InetSocketAddress (interfaces.GetAddress (size-1,0), 8080);
source->Connect (remote);

Simulator::Schedule (Seconds (1), &GenerateTraffic, source, packetSize, totalPackets,
interPacketInterval);

```

```

//std::cout << "Starting simulation for " << totalTime << " s ...\n";
AnimationInterface anim ("scratch/aodv_output.xml");

/*Ptr<FlowMonitor> flowmon;
FlowMonitorHelper flowmonHelper;
flowmon = flowmonHelper.InstallAll ();*/
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();

Simulator::Stop (Seconds (totalTime));
    //to get ascii tracing
    if(asciitracing==true)
    {
        AsciiTraceHelper ascii;
        MobilityHelper::EnableAsciiAll(ascii.CreateFileStream("iitp-trace.tr"));
        wifiPhy.EnableAsciiAll(ascii.CreateFileStream("aodv.tr"));
        //
        //

        wifiPhy.EnablePcap("aodv_sec",devices.Get(0),true);
        wifiPhy.EnablePcap("aodv_sec",devices.Get(9),true);
        //wifi.EnablePcap("aodv_sec",csmaDevices.Get(0),true);
    }
Simulator::Run ();

/*flowmon->SetAttribute("DelayBinWidth", DoubleValue(0.01));
flowmon->SetAttribute("JitterBinWidth", DoubleValue(0.01));
flowmon->SetAttribute("PacketSizeBinWidth", DoubleValue(1));
flowmon->CheckForLostPackets();
flowmon->SerializeToXmlFile("scratch/aodv_flow.xml", true, true);*/
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowmon.GetClassifier());
std::map<FlowId,FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
int j =0;
float AvgThroughput =0;
Time Delay;

NS_LOG_UNCOND("-----Results output ID wise -----"<<std::endl);

for(std::map<FlowId,FlowMonitor::FlowStats>::const_iterator iter=stats.begin
();iter!=stats.end();++iter)

```

```

{
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(iter->first);
    NS_LOG_UNCOND("-----Flow Id: "<<iter->first);
    NS_LOG_UNCOND("Src  Addr  " << t.sourceAddress <<"  DST  Addresss  "<<
t.destinationAddress);
    NS_LOG_UNCOND("Sent Packets = " << iter->second.txPackets);
    NS_LOG_UNCOND("Received Packets = "<<iter->second.rxPackets);

    NS_LOG_UNCOND("Delay = "<<iter->second.delaySum);
    NS_LOG_UNCOND("Throughput      =      " <<iter->second.rxBytes*8.0      /(iter-
>second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSeconds()
)/1024<< " Kbps");// this will show the throughput on flow id

    SentPackets = SentPackets + (iter->second.txPackets);
    ReceivedPackets=ReceivedPackets +(iter->second.rxPackets);
    AvgThroughput = AvgThroughput + (iter->second.rxBytes * 8.0 / (iter-
>second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSeconds())/1024);
    // this variable hold the value of total results of the sim
    Delay=Delay+(iter->second.delaySum);

    j=j+1;
}

    AvgThroughput=AvgThroughput/j;
    NS_LOG_UNCOND("-----Total Result of the simulation-----" << std::endl);
    NS_LOG_UNCOND("Total Sent Packets = "<< SentPackets);
    NS_LOG_UNCOND("Total Received Packets = "<< ReceivedPackets);

    NS_LOG_UNCOND("Average Throughput = " << AvgThroughput);
    NS_LOG_UNCOND(" Delay = "<<Delay );

    NS_LOG_UNCOND("Total Flow Id = "<< j);
    monitor->SerializeToXmlFile("manet-routing-compare.flowmon",true,true);

Simulator::Destroy ();

/*std::cout<<"\n\n***** OUTPUT *****\n\n";
std::cout<<"Total Packets sent = "<<packetsSent<<std::endl;
std::cout<<"Total Packets received = "<<packetsReceived<<std::endl;*/
std::cout<<"Packet  delivery  ratio  =  "<<(float)(packetsReceived/packetsSent)*100<<"
%"<<std::endl;

```

