ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное
бюджетное образовательное учреждение высшего образования
«САНКТ- ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ
ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»

Мутханна А.С., Киричек Р.В.

ПАКЕТ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ СЕТЕЙ СВЯЗИ И ПЕРЕДАЧИ ДАННЫХ NETWORK SIMULATOR 3 (NS3)

Учебное пособие

Санкт-Петербург 2018

Содержание

ВВЕДЕНИЕ 4
1. ОСНОВНЫЕ СВЕДЕНИЯ ОБ NS-3 4
1.1. Имитационные модели 5
1.2. Для пользователей NS-2 6
1.3. Ресурсы 7
1.3.1 Программные системы Mercurial 7
1.3.2 Система сборки Waf 8
1.3.3 Среда разработки 8
1.3.4 Программирование сокетов 9
1.4. NetAnim 9
2. НАЧАЛО РАБОТЫ 10
2.1. Обзор 10
2.2. Установка Ns-3 11
2.3. Загрузка ns-3 с помощью Tarball 11
2.4. Использование Bake 12
3. ПОСТРОЕНИЕ NS-3 14
3.1. Построение с помощью build.py 14
3.2. Построение с помощью bake 15
3.3. Построение с помощью Waf 16
3.4. Настройка и сборка Waf 18
3.4.1. Построение профилей 19
3.4.2. Компиляторы и флаги 20
3.4.3. Установка 20
3.4.4. One Waf 21
4. ТЕСТИРОВАНИЕ NS-3 21
4.1. Запуск скрипта 22
4.2. Аргументы программы 23
4.3. Отладка 23
4.4. Рабочий каталог 24
5. ОСНОВНЫЕ АБСТРАКЦИИ 24
5.1. Создание объектов 25
6. СОЗДАНИЕ СКРИПТА 25
6.1. Модули Includes 26
6.2. Пространство имен NS-3 26
6.3. Вход в систему 27
6.4. Функция Main 27
6.5. Объекты топологии 27
6.6. Приложения 30
6.7. Анимация 32
6.8. Трассировка 32
6.9. Запуск моделирования 33
7. СОБРАННЫЙ КОД 33

7.1. Запуск скрипта 35
8. ПОСТРОЕНИЕ СЕТИ С ТОПОЛОГИЕЙ «ШИНА» 35
9. ПОСТРОЕНИЕ БЕСПРОВОДНОЙ СЕТЕВОЙ ТОПОЛОГИИ 40
10. ПРИМЕРЫ МОДЕЛИРОВАНИЯ СЕТЕЙ IP НА NS-3 45
10.1. Пример 1 46
10.2. Пример 2 48
10.3. Пример 3 54
10.4. Пример 4 57
10.5. Пример 5 60
10.6. Пример 6 63

ВВЕДЕНИЕ

Моделирование – очень важная современная технология. Она может применяться к научным, инженерным и другим областям для различных целей. Компьютерное моделирование позволяет моделировать гипотетические и реальные объекты или действия на компьютере, чтобы их можно было изучить, а также чтобы увидеть, как функционирует система в целом. Различные переменные могут использоваться для прогнозирования поведения системы.

Сетевое моделирование помогает уделять большое внимание производительности или обоснованности использования распределенного протокола или алгоритма. Более того, поскольку сетевые технологии развиваются очень быстро, то во всем процессе участвует целый ряд различных организаций, и они используют разные технологии или продукты, работающие на различном программном обеспечении в Интернете. Вот почему для моделирования сети всегда требуются открытые платформы, которые должны быть достаточно масштабируемыми, чтобы включать различные усилия и различные пакеты при симуляции всей сети целиком. Интернет также характеризуется тем, что он структурирован с помощью унифицированного сетевого стека (TCP/IP), поэтому технологии разных уровней могут быть реализованы по-разному, но с унифицированным интерфейсом между соседними уровнями. Таким образом, средства моделирования сети должны иметь возможность включать новые пакеты и работать прозрачно, не нанося ущерба существующим компонентам или пакетам.

Сетевые симуляторы используются людьми из разных областей, например, научными исследователями или промышленными разработчиками для проектирования, моделирования, проверки и анализа производительности различных сетевых протоколов. Они также могут использоваться для оценки влияния различных параметров на исследуемые протоколы. Как правило, сетевой симулятор состоит из широкого спектра сетевых технологий и протоколов и помогает пользователям строить сложные сети из базовых строительных блоков, таких как кластеры узлов и связи между ними. С их помощью можно проектировать различные топологии сети с использованием различных типов узлов, таких как конечные узлы, концентраторы, сетевые мосты, маршрутизаторы, устройства оптического канала связи и мобильные устройства.

1. ОСНОВНЫЕ СВЕДЕНИЯ ОБ NS-3

Проект NS-3, начатый в 2006 году, был разработан для обеспечения открытой, расширяемой сетевой имитационной платформы для сетевых исследований и обучения. NS-3 – это симулятор сети с дискретными

событиями. NS-3 является свободным программным обеспечением, выпускаемым под лицензией GNU GPLv2.

Основная цель проекта NS-3 – создание симулятора для IP-сетей, поддерживающего работу на уровнях 2-4 стека сетевых протоколов с дискретным механизмом обработки событий.

Проще говоря, NS-3 предоставляет модели работы сетей пакетной передачи данных, а также пользовательский имитационный движок для проведения имитационных экспериментов. Ещё одной причиной использования NS-3 является возможность проведения исследований, которые труднее или невозможно выполнить на реальных системах, изучить поведение системы в высоко контролируемой воспроизводимой среде и узнать, как работают сети. Пользователи заметят, что доступная модель, установленная в NS-3, фокусируется на моделировании работы интернет-протоколов и сетей, но NS-3 не ограничивается интернет-системами; несколько пользователей используют NS-3 для моделирования не-интернет-систем.

Для моделирования сети существуют многие инструменты моделирования. Ниже приведены некоторые отличительные особенности NS-3 в отличие от других инструментов.

- NS-3 разработан как набор библиотек, которые могут быть объединены вместе, а также с другими внешними библиотеками программного обеспечения. Хотя некоторые платформы моделирования предоставляют пользователям единую интегрированную среду графического интерфейса пользователя, в которой выполняются все задачи, NS-3 является более модульной в этом отношении. Для NS-3 можно использовать несколько внешних аниматоров и инструменты для анализа и визуализации данных. Тем не менее, пользователи должны быть готовы к работе в командной строке и с инструментами разработки программного обеспечения на языках C ++ и / или Python.
- NS-3 в основном используется в системах Linux, хотя существует поддержка FreeBSD, Cygwin (для Windows)
- NS-3 не является официально поддерживаемым программным продуктом компании. Поддержка NS-3 выполняется с максимальной эффективностью в списке рассылки ns-3-users.

1.1. Имитационные модели

Проект NS-3 стремится к созданию прочного ядра моделирования, который хорошо документирован, прост в использовании и отладке, и что позволяет удовлетворять потребности всего рабочего процесса имитации, от конфигурации моделирования до сбора и анализа информации.

Кроме того, программное обеспечение инфраструктуры NS-3 поддерживает разработку имитационных моделей, которые являются

достаточно реалистичными, чтобы позволить NS-3 для использования в качестве эмулятора сети в реальном времени. Ядро моделирования ns-3 поддерживает исследования как в IP, так и в не-IP сетях. Однако большая часть пользователей сосредоточена на имитации беспроводных / IP-сетей, которые включают модели для Wi-Fi, WiMAX или LTE для 1-ого и 2-ого уровней и различные статические или динамические протоколы маршрутизации, такие как OLSR и AODV для IP-приложений.

NS-3 также поддерживает в режиме реального времени планировщик, который облегчает использование сценариев "моделирование в цикле" для взаимодействия с реальными системами. Например, пользователи могут отправлять и принимать NS-3-сгенерированные пакеты на реальных сетевых устройствах, и NS-3 может служить в качестве основы взаимосвязи, чтобы добавить эффекты соединения между виртуальными машинами.

Еще один акцент в симуляторе - повторное использование реального приложения и кода ядра. В настоящее время тестируются и оцениваются платформы для запуска не модифицированных приложений или всего сетевого ядра ядра Linux в NS-3.

1.2. Для пользователей NS-2

Для тех, кто знаком с ns-2 (популярным инструментом, который предшествовал ns-3), наиболее заметным внешним изменением при переходе на ns-3 является выбор языка сценариев. Программы в ns-2 создаются по сценарию в OTcl, и результаты имитаций можно визуализировать с помощью Network Animator nam. Невозможно запустить моделирование в ns-2 чисто из C ++ (т.е. в качестве основной программы без какого-либо OTcl). Более того, некоторые компоненты ns-2 написаны на C ++, а другие - в OTcl. В ns-3 симулятор написан полностью на C ++, с дополнительными привязками Python. Поэтому сценарии моделирования могут быть написаны на C ++ или в Python. Поскольку ns-3 генерирует файлы трассировки пакетов «pcap», другие утилиты также могут использоваться для анализа трасс.

Некоторые ns-2 модели, которые были написаны на C++, уже были спортированы на ns-3. Основанные на OTcl модели не могут быть перенесены “как есть” и требую повторного написания.

Отличительные черты между ns-2 и ns-3:

• Ns-3 активно поддерживается активным, отзывчивым списком рассылки пользователей, а ns-2 поддерживается лишь незначительно и в течение десятилетия не наблюдала значительного развития в своем основном дереве кода.

• Ns-3 предоставляет функции, недоступные в ns-2, такие как среда выполнения кода реализации (позволяющая пользователям запускать реальный код реализации в симуляторе).

• Ns-3 обеспечивает более низкий базовый уровень абстракции по сравнению с ns-2, что позволяет лучше согласоваться с тем, как складываются реальные системы. Некоторые ограничения, найденные в ns-2 (например, поддержка нескольких типов интерфейсов на узлах правильно) были устранены в ns-3.

Ns-2 имеет более разнообразный набор дополнительных модулей, чем ns-3, благодаря своей долгой истории. Тем не менее ns-3 имеет более подробные модели в нескольких популярных областях исследований (включая сложные модели LTE и WiFi), а его поддержка кода реализации допускает очень широкий спектр моделей высокой точности. Пользователи могут быть удивлены, узнав, что весь сетевой стек Linux можно инкапсулировать в узел ns-3, используя структуру Direct Code Execution (DCE).

1.3. Ресурсы

Существует несколько важных ресурсов, о которых должен знать любой пользователь ns-3. Основной веб-сайт находится по адресу http://www.nsnam.org и предоставляет доступ к основной информации о системе ns-3. Подробная документация доступна на основном веб-сайте по адресу http://www.nsnam.org/documentation/. С этой страницы также можно найти документы, относящиеся к архитектуре системы.

Существует вики, которая дополняет основной веб-сайт ns-3, который вы найдете на http://www.nsnam.org/wiki/. Здесь вы найдете ответы на часто задаваемые вопросы пользователей и разработчиков, а также руководства по поиску неисправностей, сторонний код, документы и т. д.

Исходный код можно найти и просмотреть на сайте http://code.nsnam.org/. Там вы найдете текущее дерево разработки в репозитории с именем *ns-3-dev*. Здесь также можно найти последние релизы и экспериментальные хранилища основных разработчиков.

1.3.1 Программные системы Mercurial

Комплексные программные системы нуждаются в некотором способе управления организацией и изменениях в базовом коде и документировании. Есть много способов выполнить это действие, и вы, возможно, слышали о некоторых системах, которые в настоящее время используются для этого. Concurrent Version System (CVS), вероятно, наиболее известна.

Проект ns-3 использует Mercurial в качестве системы управления исходным кодом. Mercurial имеет веб-сайт по адресу

http://www.selenic.com/mercurial/, из которого вы можете получить исходные или последующие версии этой системы управления конфигурацией программного обеспечения.

1.3.2 Система сборки Waf

После того, как исходный код был загружен в вашу локальную систему, вам необходимо скомпилировать этот источник для создания пригодных к использованию программ. Как и в случае с управлением исходным кодом, для выполнения этой функции имеется много инструментов. Вероятно, самым известным из этих инструментов является *make*. Наряду с самым известным инструментом, сделать, что то похожее, и использовать в очень большой и настраиваемой системе наиболее трудно. В связи с этим было разработано много альтернатив. Недавно эти системы были разработаны с использованием языка Python.

Система сборки Waf используется в проекте ns-3. Это одно из новых поколений, построенных на Python систем. Вам не нужно понимать какой-либо Python для построения существующей системы ns-3.

Для тех, кто интересует более подробно Waf, главный веб-сайт можно найти на странице http://code.google.com/p/waf/.

1.3.3 Среда разработки

Как упоминалось выше, сценарии в ns-3 выполняются на C ++ или Python. Большая часть API для ns-3 доступна в Python, но модели написаны на C ++ в любом случае. Для работы с NS -3 необходимо знание C ++ и объектно-ориентированных понятий.

Если вы новичок в C ++, вы можете найти учебник или веб-сайт и проработать, по крайней мере, основные функции языка перед продолжением знакомства с NS-3.

Система ns-3 использует несколько компонентов GNU "toolchain" для разработки. Программная инструментальная цепь - это набор инструментов программирования, доступных в данной среде. Для быстрого обзора того, что включено в состав набора инструментов GNU, см. http://en.wikipedia.org/wiki/GNU_toolchain. Ns-3 использует gcc, GNU binutils и gdb. Однако мы не используем инструменты сборки GNU, ни make, ни autotools. Мы используем Waf для этих функций.

Обычно для работы в ns-3 используется Linux. Для тех, кто работает под Windows, существуют среды, которые в разной степени имитируют среду Linux. Проект ns-3 в прошлом (но не сейчас) поддерживал разработку в среде Cygwin для этих пользователей. См. http://www.cygwin.com/ для получения дополнительной информации о загрузке и посетите wiki для ns-3 для получения дополнительной информации о Cygwin и ns-3. В настоящее время MinGW официально не поддерживается. Еще одной альтернативой Cygwin является установка

среды виртуальной машины, такой как сервер VMware, и установка виртуальной машины Linux.

1.3.4 Программирование сокетов

Мы предположим, что в примерах, используемых в этом учебном пособии, базовый объект с API-интерфейсом Berkeley Sockets API. Если вы новичок в сокетах, рекомендуется ознакомиться с API и некоторыми распространенными вариантами использования. Для хорошего обзора программирования сокетов TCP / IP мы рекомендуем сокеты TCP / IP на C, Donahoo и Calvert.

Существует связанный веб-сайт, содержащий источник примеров в книге, которые вы можете найти по адресу: http://cs.baylor.edu/~donahoo/practical/CSockets/.

1.4. NetAnim

NetAnim – представляет собой программный инструмент, основанный на наборе инструментов Qt 4. Он имитирует симуляцию, используя файл трассировки XML, собранный во время моделирования, для графического вывода.

Обзор возможностей:

- Анимация пакетов по проводным и беспроводным каналам
- Временная шкала пакетов с использованием фильтра регулярных выражений для метаданных пакета.
- Статистика положения узла с графиком построения траектории узла (путь мобильного узла).

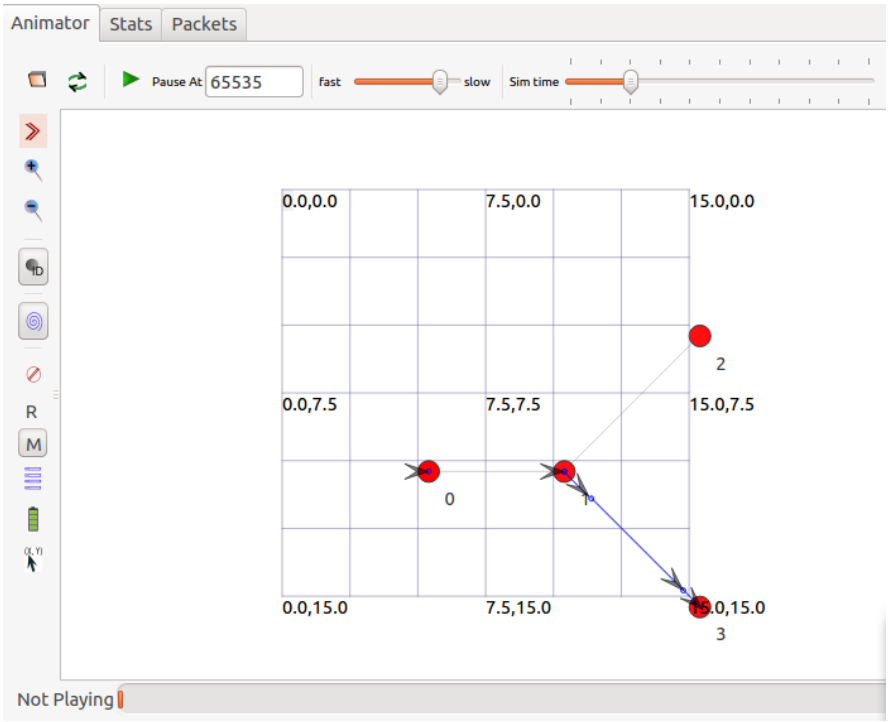

Рис.1. Пример возможностей NetAnim.

2. НАЧАЛО РАБОТЫ

Этот раздел предназначен для введения пользователя в работу, начиная с компьютера, на котором никогда не был установлен ns-3. В нем рассматриваются поддерживаемые платформы, предварительные условия, способы получения ns-3, способы создания ns-3 и способы проверки вашей сборки и запуска простых программ.

2.1. Обзор

Ns-3 построен как система программных библиотек, которые работают вместе. Пользовательские программы будут написаны из этих библиотек. Пользовательские программы написаны на языках программирования C ++ или Python.

Ns-3 распространяется как исходный код, что означает, что целевой системе необходимо иметь среду разработки программного обеспечения, чтобы сначала создать библиотеки, а затем создать пользовательскую программу. Ns-3 может быть распределен в виде готовых библиотек для выбранных систем и таким же образом он может быть распределен в будущем, но в настоящее время многие пользователи фактически выполняют свою работу, редактируя ns-3, имея исходный код, чтобы перестроить библиотеки. Если кто-то захочет взять на себя работу по созданию готовых библиотек и пакетов для операционных систем, обратитесь к списку рассылки *ns-developers*.

Далее мы рассмотрим два способа загрузки и сборки ns-3:

Первый - загрузка и установка официального релиза с основного веб-сайта.

Второй - выборка и создание копий для разработки ns-3.

Мы рассмотрим оба примера, поскольку используемые инструменты немного отличаются.

2.2. Установка Ns-3

Система ns-3 в целом представляет собой довольно сложную систему и имеет ряд зависимостей от других компонентов. Наряду с системами, с которыми вы каждый день будете иметь дело (набор инструментов GNU, Mercurial, текстовый редактор), вам необходимо убедиться, что перед вашей работой в вашей системе имеется несколько дополнительных библиотек.

Предположим, что загружаем NS-3 на Linux и имеет набор инструментов GNU. Также предположим, что уже установлен и запущен Mercurial и Waf в целевой системе.

Код ns-3 доступен в репозиториях Mercurial на сервере http://code.nsnam.org. Вы также можете скачать релиз tarball по адресу http://www.nsnam.org/release/, или вы можете работать с репозиториями, используя Mercurial. Рекомендуется использовать Mercurial.

Самый простой способ начать использовать репозитории Mercurial - использовать среду *ns-3-allinone*. Это набор скриптов, который управляет загрузкой и созданием для вас различных подсистем ns-3.

Одна из практик - создать каталог, называемый рабочим пространством, в своем домашнем каталоге, в котором можно хранить локальные репозитории Mercurial. Можно задать любое имя каталога, но репозитории могут также использоваться в некоторой документации в качестве примера имени каталога.

2.3. Загрузка ns-3 с помощью Tarball

Tarball - это особый формат архива программного обеспечения, в котором несколько файлов объединяются вместе, а архив может быть сжат. Программные релизы ns-3 предоставляются через загружаемый архив. Процесс загрузки ns-3 с помощью tarball прост; Вам просто нужно выбрать выпуск, загрузить его и распаковать.

Предположим, что вы хотите построить ns-3 в локальном каталоге, называемом workspace. Если вы примете подход к каталогу рабочей области, вы можете получить копию релиза, введя следующую строку в свою оболочку Linux (разумеется, замените соответствующие номера версий):

```
$ cd
$ mkdir workspace
$ cd workspace
```

```
$ wget http://www.nsnam.org/release/ns-allinone-3.26.tar.bz2
$ tar xjf ns-allinone-3.26.tar.bz2
```

Если вы перейдете в каталог ns-allinone-3.26, вы увидите несколько файлов и каталогов:

```
$ ls
bake      constants.py   ns-3.26                                  README
build.py  netanim-3.107  pybindgen-0.17.0.post57+nga6376f2       util.py
```

Теперь все готово к созданию базового дистрибутива ns-3 и можете перейти к разделу по сборке ns-3.

2.4. Использование Bake

Bake - это инструмент для распределенной интеграции и построения, разработанный для проекта ns-3. Bake можно использовать для извлечения версий разработки программного обеспечения ns-3, а также для загрузки и сборки расширений к базовому распределению ns-3, таких как среда Direct Code Execution, Network Simulation Cradle, возможность создавать новые привязки Python и другие.

В последних версиях ns-3, Bake был включен в релиз tarball. Файл конфигурации, включенный в выпущенную версию, позволит загружать любое программное обеспечение, которое было актуальным на момент выпуска. То есть, например, версия Bake, распространяемая вместе с выпуском ns-3.21, может быть использована для извлечения компонентов для этой версии ns-3 или более ранней версии, но не может использоваться для извлечения компонентов для последующих выпусков (если только *bakeconf .xml* файл обновлен).

Вы также можете получить самую последнюю копию *bake*, введя следующую строку в оболочку Linux (при условии, что вы установили Mercurial):

```
$ cd
$ mkdir workspace
$ cd workspace
$ hg clone http://code.nsnam.org/bake
```

В ходе выполнения команды *hg* (Mercurial), вы увидите что-то подобное:

```
...
destination directory: bake
requesting all changes
adding changesets
adding manifests
adding file changes
added 339 changesets with 796 changes to 63 files
updating to branch default
45 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

После завершения команды *clone* у вас должен быть каталог с именем *bake*, содержимое которого должно выглядеть примерно так:

```
$ ls
bake          bakeconf.xml      doc      generate-binary.py        TODO
```

```
bake.py       examples          test
```

Обратите внимание на то, что вы только что загрузили скрипты Python и модуль Python, называемый *bake*. Следующим шагом будет использование этих сценариев для загрузки и сборки дистрибутива ns-3 по вашему выбору.

Существует несколько доступных конфигурационных целей:

1. *ns-3.26*: модуль, соответствующий релизу; он загрузит компоненты, похожие на архив tar.
2. *ns-3-dev*: аналогичный модуль, но с использованием дерева кодов разработки.
3. *ns-allinone-3.26*: модуль, включающий другие дополнительные функции, такие как click routing, openflow для ns-3 и Network Simulation Cradle.
4. *ns-3-allinone*: аналогично выпущенной версии модуля allinone, но для кода разработки.

Теперь используем инструмент *bake*.

bake работает, загружая исходные пакеты в исходный каталог и установив библиотеки в каталог сборки. bake можно запустить, обратившись к двоичному файлу, но, если вы решите запустить bake из-за пределов каталога, в который он был загружен, рекомендуется положить его в ваш путь, например, ниже (пример оболочки bash для Linux). Сначала перейдите в каталог «bake», а затем установите следующие переменные среды:

```
$ export BAKE_HOME=`pwd`
$ export PATH=$PATH:$BAKE_HOME:$BAKE_HOME/build/bin
$ export PYTHONPATH=$PYTHONPATH:$BAKE_HOME:$BAKE_HOME/build/lib
```

Это приведет программу bake.py в путь оболочки и позволит другим программам находить исполняемые файлы и библиотеки, созданные методом *bake*. Хотя в некоторых случаях использования *bake* не требуется установка PATH и PYTHONPATH, как описано выше, обычно выполняются полные сборки ns-3-allinone (с дополнительными пакетами).

Переходим в каталог рабочей области и вводим в строку следующее:

```
$ ./bake.py configure -e ns-3.26
```

Далее мы попросим *bake* проверку наличия у нас достаточного количества инструментов для загрузки различных компонентов:

```
$ ./bake.py check
```

Вы должны увидеть следующее:

```
> Python - OK
> GNU C++ compiler - OK
> Mercurial - OK
> CVS - OK
> GIT - OK
> Bazaar - OK
> Tar tool - OK
> Unzip tool - OK
> Unrar tool - is missing
> 7z data compression utility - OK
> XZ data compression utility - OK
```

```
> Make - OK
> cMake - OK
> patch tool - OK
> autoreconf tool - OK
> Path searched for tools: /usr/lib64/qt-3.3/bin /usr/lib64/ccache
/usr/local/bin /bin /usr/bin /usr/local/sbin /usr/sbin /sbin
/home/tomh/bin bin
```

Для дальнейшей работы должны быть установлены следующие инструменты загрузки: Mercurial, CVS, GIT и Bazaar, поскольку они позволяют нам получить код.

Затем пробуем загрузить программное обеспечение:

```
$ ./bake.py download
```

Должно получиться следующее

```
>> Downloading gccxml-ns3 (target directory:gccxml) - OK
>> Searching for system dependency python-dev - OK
>> Searching for system dependency pygraphviz - OK
>> Searching for system dependency pygoocanvas - OK
>> Searching for system dependency setuptools - OK
>> Searching for system dependency g++ - OK
>> Searching for system dependency qt4 - OK
>> Downloading pygccxml - OK
>> Downloading netanim-3.107 - OK
>> Downloading pybindgen-0.17.0.post57+nga6376f2 (target
directory:pybindgen) - OK
>> Downloading ns-3.26 – OK
```

Это говорит о том, что было загружено пять источников. Проверьте исходный каталог, для этого введите *ls*:

```
$ ls
gccxml   netanim-3.107   ns-3.26   pybindgen   pygccxml   pygccxml-1.0.0.zip
```

Теперь можно приступать сборке ns-3.

3. ПОСТРОЕНИЕ NS-3

3.1. Построение с помощью build.py

При работе с выпущенным tarball при первом запуске проекта ns-3 вы можете построить его с помощью удобной программы из каталога allinone. Эта программа называется *build.py*. Эта программа получит проект, сконфигурированный для вас наиболее полезным способом. Однако учтите, что более сложная конфигурация и работа с ns-3, как правило, предполагает использование встроенной системы сборки ns-3 Waf, которая будет представлена далее.

Если вы загрузили с помощью tarball, у вас должен быть каталог, называемый *ns-allinone-3.26* в каталоге *~ / workspace*. Введите следующее:

```
$ ./build.py --enable-examples --enable-tests
```

По умолчанию программа также создает все доступные модули. Позже вы можете построить ns-3 без примеров и тестов или устранить модули, которые не нужны для вашей работы. Вы увидите много

типичных сообщений о выходе компилятора, отображаемых в виде скрипта сборки, который собирает различные загруженные вами фрагменты. В итоге вы должны увидеть следующее:

```
Waf: Leaving directory `/path/to/workspace/ns-allinone-3.26/ns-3.26/build'
'build' finished successfully (6m25.032s)
Modules built:
antenna                  aodv                     applications
bridge                   buildings                config-store
core                     csma                     csma-layout
dsdv                     dsr                      energy
fd-net-device            flow-monitor             internet
internet-apps            lr-wpan                  lte
mesh                     mobility                 mpi
netanim (no Python)      network                  nix-vector-routing
olsr                     openflow (no Python)     point-to-point
point-to-point-layout    propagation              sixlowpan
spectrum                 stats                    tap-bridge
test (no Python)         topology-read            traffic-control
uan                      virtual-net-device       visualizer
wave                     wifi                     wimax


Modules not built (see ns-3 tutorial for explanation):
brite click
```

Относительно части о нестроенных модулях:

```
Modules not built (see ns-3 tutorial for explanation):
brite                     click
```

Это означает, что некоторые модули ns-3, которые имеют зависимости от внешних библиотек, возможно, не были созданы, или что в конфигурации специально попросили не создавать их. Это не означает, что симулятор не построился успешно или что он обеспечит неправильные результаты для модулей, перечисленных в качестве встроенных.

3.2. Построение с помощью bake

Если вы использовали *bake* для извлечения исходного кода из репозиториев проекта, вы можете продолжать использовать его для сборки ns-3.

```
$ ./bake.py build
```

И вы увидите следующее:

```
>> Building gccxml-ns3 - OK
>> Building pygccxml - OK
>> Building netanim-3.107 - OK
>> Building pybindgen-0.17.0.post57+nga6376f2 - OK
>> Building ns-3.26 - OK
```

Подсказка: вы также можете выполнить оба этапа, загрузить и создать, написав команду" bake.py deploy ".

Если случился сбой, посмотрите, что говорит следующая команда: Она может дать подсказку относительно отсутствующей зависимости:

```
$ ./bake.py show
```

Это перечислит различные зависимости пакетов, которые вы пытаетесь построить.

3.3. Построение с помощью Waf

До этого момента мы использовали скрипт *build.py* или инструмент *bake* , чтобы приступить к созданию ns-3. Эти инструменты полезны для построения ns-3 и поддерживающих библиотек, и они обращаются в каталог ns-3. Теперь с помощью инструмент сборки Waf соберем наш проект. Большинство пользователей быстро переходят к использованию Waf напрямую для настройки и сборки ns-3. Чтобы продолжить, нужно сменить ваш рабочий каталог на ns-3 каталог, который вы изначально построили.

Сейчас это не является обязательным, но будет очень полезно повторить и посмотреть, как внести изменения в конфигурацию проекта. Вероятно, наиболее полезным изменением конфигурации, которое вы можете сделать, будет построение оптимизированной версии кода. По умолчанию вы сконфигурировали проект для построения отладочной версии. Сделаем оптимизированную сборку, которые включают примеры и тесты, вам нужно будет выполнить следующие команды:

```
$ ./waf clean
$ ./waf configure --build-profile=optimized --enable-examples --enable-tests
```

Первая команда для очистки предыдущей сборки обычно не является строго необходимой, но является хорошей практикой. Он удалит ранее созданные библиотеки и объектные файлы, найденные в каталоге *build /*. Когда проект перенастроен и система сборки проверяет различные зависимости, вы увидите такой результат:

```
Setting top to                                    :.
Setting out to                                    : build
Checking for 'gcc' (c compiler)                   : /usr/bin/gcc
Checking for cc version                           : 4.2.1
Checking for 'g++' (c++ compiler)                 : /usr/bin/g++
Checking boost includes                           : 1_46_1
Checking boost libs                               : ok
Checking for boost linkage                        : ok
Checking for click location                       : not found
Checking for program pkg-config                   : /sw/bin/pkg-config
Checking for 'gtk+-2.0' >= 2.12                   : yes
Checking for 'libxml-2.0' >= 2.7                  : yes
Checking for type uint128_t                       : not found
Checking for type __uint128_t                     : yes
Checking high precision implementation            : 128-bit integer (default)
Checking for header stdint.h                      : yes
Checking for header inttypes.h                    : yes
Checking for header sys/inttypes.h                : not found
Checking for header sys/types.h                   : yes
Checking for header dirent.h                      : yes
Checking for header sys/stat.h                    : yes
Checking for header stdlib.h                      : yes
Checking for header signal.h                     Checking for header stdint.h
```

```
Checking for header pthread.h                  : yes
Checking for header inttypes.h                 : yes
: yes                                          : yes
Checking for header sys/inttypes.h             : not found
Checking for library rt                        : not found
Checking for header netpacket/packet.h         : not found
Checking for header sys/ioctl.h                : yes
Checking for header net/if.h                   : not found
Checking for header net/ethernet.h             : yes
Checking for header linux/if_tun.h             : not found
Checking for header netpacket/packet.h         : not found
Checking for NSC location                      : not found
Checking for 'mpic++'                          : yes
Checking for 'sqlite3'                         : yes
Checking for header linux/if_tun.h             : not found
Checking for program sudo                      : /usr/bin/sudo
Checking for program valgrind                  : /sw/bin/valgrind
Checking for 'gsl'                             : yes
Checking for compilation flag -Wno-error=deprecated-d... support    : ok
Checking for compilation flag -Wno-error=deprecated-d... support    : ok
Checking for compilation flag -fstrict-aliasing... support          : ok
Checking for compilation flag -fstrict-aliasing... support          : ok
Checking for compilation flag -Wstrict-aliasing... support          : ok
Checking for compilation flag -Wstrict-aliasing... support          : ok
Checking for program doxygen                   : /usr/local/bin/doxygen
---- Summary of optional NS-3 features:
Build profile           : debug
BRITE Integration       : not enabled (BRITE not enabled (see option --with-
                          brite))
Build directory                 : build
Build examples                  : enabled
Build tests                     : enable
Emulated Net Device       :enabled(<netpacket/packet.h> include not
                           detected)
Emulation FdNetDevice     : not enabled (needs netpacket/packet.h)
File descriptor NetDevice       : enabled
GNU Scientific Library (GSL)    : enabled
GtkConfigStore                  : enabled
MPI Support                     : enabled
NS-3 Click Integration          : not enabled (nsclick not enabled (see
                                  option --with-nsclick))
NS-3 OpenFlow Integration       : not enabled (Required boost libraries not
                                  found, missing: system, signals, filesystem)
Network Simulation Cradle       : not enabled (NSC not found (see option --
                                  with-nsc))
PlanetLab FdNetDevice           : not enabled (PlanetLab operating system
                                  not detected (see option --force-planetlab))
PyViz visualizer                : enabled
Python Bindings                 : enabled
Real Time Simulator             : enabled (librt is not available)
SQlite stats data output        : enabled
Tap Bridge            : not enabled (<linux/if_tun.h> include not detected)
Tap FdNetDevice       : not enabled (needs linux/if_tun.h)
Threading Primitives            : enabled
Use sudo to set suid bit        : not enabled (option --enable-sudo not
                                  selected)
XmlIo                           : enabled
'configure' finished successfully (1.944s)
```

Обратите внимание на последнюю часть вышеприведенного вывода. Некоторые параметры ns-3 не включены по умолчанию или требуют

поддержки от версии для правильной работы. Например, чтобы включить XmlTo, в системе должна быть найдена библиотека libxml-2.0. Если эта библиотека не была найдена, соответствующая функция ns-3 не будет включена и будет отображаться сообщение. Обратите также внимание на то, что есть возможность использовать программу *sudo* для установки бита suid для определенных программ. По умолчанию эта функция отключена, поэтому эта функция отображается как «не включена». Наконец, чтобы распечатать эту сводку о том, какие дополнительные функции включены, используйте параметр *--check-config* для waf.

Теперь вернемся к сборке отладки, которая включает примеры и тесты.

```
$ ./waf clean
$ ./waf configure --build-profile = debug --enable-examples --enable-tests
```

Теперь система сборки настроена, и вы можете создавать отладочные версии программ ns-3, просто введя:

```
$ ./waf
```

Существует команда для проверки того, какой профиль активен в настоящее время для уже настроенного проекта:

```
$ ./waf --check-profile
Waf: Entering directory \`/path/to/ns-3-allinone/ns-3.26/build'
Build profile: debug
```

Сценарий *build.py*, рассмотренный выше, также поддерживает аргументы *--enable-examples* и *enable-tests*, но, в общем, не поддерживает напрямую другие опции waf; например, если это не сработает:

```
$ ./build.py --disable-python
```

Приведет к сообщению:

```
build.py: error: no такой опции: --disable-python.
```

Однако специальный оператор - может использоваться для передачи дополнительных опций до waf, поэтому вместо вышенаписанного будет работать:

```
$ ./build.py -- --disable-python,
```

так как он генерирует базовую команду *./waf configure --disable-python*.

3.4. Настройка и сборка Waf

Некоторые команды Waf имеют смысл только на этапе настройки, а некоторые команды действительны на этапе сборки. Например, если вы хотите использовать функции эмуляции ns-3, вы можете установить бит *suid*, используя *sudo*, как описано выше. Это будет команда временной конфигурации, и вы можете перенастроить ее с помощью следующей команды, которая также включает примеры и тесты.

```
$ ./waf configure --enable-sudo --enable-examples --enable-tests
```

Если вы это сделаете, Waf запустит *sudo*, чтобы изменить программы создателя сокета в коде эмуляции и запустить его как *root*.

В Waf есть множество других опций настройки и сборки. Чтобы изучить эти параметры, введите:

```
$ ./waf --help.
```

Мы будем использовать некоторые из команд, связанных с тестированием, в следующем разделе.

3.4.1. Построение профилей

Вы можете настроить Waf для отладки или оптимизированных сборок:

```
$ ./waf --build-profile=debug
```

Существует также промежуточный профиль сборки, *release.* *-d* является синонимом для *-build-profile*.

Профиль сборки контролирует использование протоколирования, утверждений и оптимизации компилятора:

Feature	Build Profile		
	debug	release	optimized
Enabled Features	NS_BUILD_PROFILE_DEBUG NS_LOG… NS_ASSERT…	NS_BUILD_PROFILE_REALEASE	NS_BUILD_PROFILE_0TIMIZED
Code Wrapper Macro	NS_BUILD_DEBUG(code)	NS_BUILD_RELEASE(code)	NS_BUILD_OPTEMIZED(code)
Complier Flags	-00 -ggdb –g3	-03 –g0 -fomit-frame-pointer	-03 –g -fstrict-overflow -march-native

Табл.1. Профили сборки

Как видите, протоколирование и утверждения доступны только в отладочных сборках. Рекомендуемой практикой является разработка вашего сценария в режиме отладки, а затем повторное выполнение (для статистики или изменение параметров) в оптимизированном профиле сборки.

Если у вас есть код, который должен запускаться только в определенных профилях сборки, используйте указанный макрос Code Wrapper:

```
NS_BUILD_DEBUG (std::cout << "Part of an output line..." << std::flush;
timer.Start ());
DoLongInvolvedComputation ();
NS_BUILD_DEBUG (timer.Stop (); std::cout << "Done: " << timer << std::endl;)
```

По умолчанию Waf помещает артефакты сборки в каталог сборки. Вы можете указать другой выходной каталог с опцией *--out*, например.

```
$ ./waf configure --out=foo
```

Сочетание этого с профилями сборки позволяет вам переключаться между различными параметрами компиляции в чистом виде:

```
$ ./waf configure --build-profile=debug --out=build/debug
$ ./waf build
...
$ ./waf configure --build-profile=optimized --out=build/optimized
$ ./waf build
...
```

Это позволяет работать с несколькими сборками, а не перезаписывать последнюю сборку. Когда вы переключаетесь, Waf будет компилировать только то, что ему нужно, вместо того, чтобы перекомпилировать все.

Когда вы, таким образом, переключаете профили сборки, то должны быть осторожны, чтобы каждый раз давать одни и те же параметры конфигурации. Удобно определить некоторые переменные среды, которые помогут вам избежать ошибок:

```
$ export NS3CONFIG="--enable-examples --enable-tests"
$ export NS3DEBUG="--build-profile=debug --out=build/debug"
$ export NS3OPT=="--build-profile=optimized --out=build/optimized"
$ ./waf configure $NS3CONFIG $NS3DEBUG
$ ./waf build
...
$ ./waf configure $NS3CONFIG $NS3OPT
$ ./waf build
```

3.4.2. Компиляторы и флаги

В приведенных выше примерах Waf использует компилятор GCC C ++, g ++, для построения ns-3. Тем не менее, можно изменить компилятор C ++, используемый Waf, указав переменную среды *CXX*. Например, чтобы использовать компилятор Clang C ++, clang ++

```
$ CXX="clang++" ./waf configure
$ ./waf build
```

Можно также настроить Waf для распределенной компиляции с *distcc* аналогичным образом:

```
$ CXX="distcc g ++" ./waf configure
$ ./waf build
```

Чтобы добавить флаги компилятора, используйте переменную среды CXXFLAGS_EXTRA при настройке ns-3.

3.4.3. Установка

Waf можно использовать для установки библиотек в разных местах системы. Место по умолчанию, где размещаются библиотеки и исполняемые файлы, находится в каталоге сборки, и поскольку Waf знает расположение этих библиотек и исполняемых файлов, нет необходимости устанавливать библиотеки в другом месте.

Если пользователи предпочитают устанавливать объекты вне каталога сборки, пользователи могут набрать *./waf install command*. По умолчанию префиксом для установки является */usr/local*, поэтому *./waf install* будет устанавливать программы в */usr /local/bin*, библиотеки в */usr/local/lib* и заголовки в */usr /local/include*. Привилегии суперпользователя обычно необходимы для установки в префикс по умолчанию, поэтому типичной командой будет *sudo ./waf install*. При запуске программ с Waf, Waf сначала предпочитает использовать разделяемые библиотеки в каталоге сборки, а затем будет искать библиотеки в пути библиотеки, сконфигурированном в локальной среде. Поэтому при установке

библиотек в систему рекомендуется проверять, используются ли используемые библиотеки.

Пользователи могут выбрать установку в другой префикс, передав опцию *--prefix* во время настройки, например:

```
./waf configure --prefix = / opt / local.
```

Если позже после сборки пользователь выдает команду *./waf install*, Префикс */ opt / local* будет использоваться.

Команда *./waf clean* должна быть использована до перенастройки проекта, если Waf будет использоваться для установки вещей с другим префиксом.

Таким образом, нет необходимости вызывать *./waf install* для использования ns-3. Большинство пользователей не будут нуждаться в этой команде, так как Waf возьмет текущие библиотеки из каталога сборки, но некоторые пользователи могут счесть это полезным, если их использование связано с работой с программами вне каталога ns-3.

3.4.4. One Waf

Существует только один скрипт Waf на верхнем уровне исходного дерева ns-3. При работе, вы можете потратить много времени с *scratch/* или с *src / ...,* и вам нужно будет вызвать Waf. Вы можете просто запомнить ваше положение, и вызвать Waf следующим образом:

```
$ ../../../waf ...
```

Но это утомительно и подверженно ошибкам и есть решения лучше. Если у вас есть полный репозиторий ns-3, то начинаем так:

```
$ cd $ (hg root) && ./waf ...
```

Еще лучше определить это как функцию оболочки:

```
$ function waff { cd $ (hg root) && ./waf $*; }
$ waff build
```

Если у вас есть только tarball, может помочь переменная окружения:

```
$ export NS3DIR = "$PWD"
$ function waff {cd $NS3DIR && ./waf $*;  }
$ cd scratch
$ waff build
```

4. ТЕСТИРОВАНИЕ NS-3

Вы можете запустить модульные тесты для распространения ns-3, запустив основной скрипт *./test.py -c*:

```
$ ./test.py -c core
```

Эти тесты запускаются параллельно с помощью Waf. Вы должны увидеть отчет о том, что

```
92 of 92 tests passed (92 passed, 0 failed, 0 crashed, 0 valgrind errors)
```

Это важное сообщение.

Вы также увидите итоговый вывод от Waf и тестового бегуна, выполняющего каждый тест, который на самом деле будет выглядеть примерно так:

```
Waf: Entering directory `/path/to/workspace/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/path/to/workspace/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.799s)
Modules built:
aodv                         applications               bridge
click                        config-store               core
csma                         csma-layout                dsdv
emu                          energy                      flow-monitor
internet                     lte                         mesh
mobility                     mpi                         netanim
network                      nix-vector-routing         ns3tcp
ns3wifi                      olsr                        openflow
point-to-point               point-to-point-layout      propagation
spectrum                     stats                      tap-bridge
template                     test                       tools
topology-read                uan                        virtual-net-device
visualizer                   wifi                       wimax
PASS: TestSuite ns3-wifi-interference
PASS: TestSuite histogram
...
PASS: TestSuite object
PASS: TestSuite random-number-generators
92 of 92 tests passed (92 passed, 0 failed, 0 crashed, 0 valgrind errors)
```

Эта команда обычно запускается пользователями, чтобы быстро проверить правильность сборки дистрибутива ns-3. (Обратите внимание, что порядок строк *PASS: ...* может меняться. Важно, чтобы итоговая строка в конце отчета сообщала, что все тесты прошли, ни один из них не завершился неудачей или разбился.)

4.1. Запуск скрипта

Запускаем скрипты под управлением Waf. Это позволяет системе сборки гарантировать, что пути разделяемой библиотеки установлены правильно и что библиотеки доступны во время выполнения. Для запуска программы просто используйте опцию *--run* в Waf. Запустим, эквивалент ns-3 универсальной программы, например, *hello world,* набрав следующее:

```
$ ./waf --run hello-simulator
```

Сначала Waf проверяет, правильно ли построена программа и выполняет сборку, если это необходимо. Затем Waf выполняет программу, которая выдает следующий результат.

```
Hello Simulator
```

Теперь вы пользователь ns-3!

Если вы видите сообщения Waf, указывающие на то, что сборка была успешно завершена, но не видите результат «*Hello Simulator*», скорее всего, вы переключили свой режим сборки на оптимизацию в разделе «*Building with Waf*», но пропустили изменения обратно в режим *debug*. Весь вывод

консоли использует специальный компонент регистрации ns-3, который полезен для вывода сообщений пользователя на консоль. Вывод этого компонента автоматически отключается при компиляции оптимизированного кода - он «*optimized out*». Если вы не видите вывод «*Hello Simulator*», введите следующее:

```
$ ./waf configure --build-profile=debug --enable-examples --enable-tests
```

чтобы указать Waf для создания отладочных версий программ ns-3, которые включают примеры и тесты. Вы должны создать фактическую версию отладки кода, введя

```
$ ./waf
```

Теперь, если вы запустите программу *hello-simulator*, вы должны увидеть нужный результат.

4.2. Аргументы программы

Чтобы передать аргументы командной строки программе ns-3, используйте этот шаблон:

```
$ ./waf --run <ns3-program> --command-template="%s <args>"
```

Замените имя своей программы на <*ns3-program*> и аргументы для <*args*>. Аргумент *--command-template* для Waf является в основном рецептом для построения фактической командной строки, которую Waf должен использовать для выполнения программы. Waf проверяет, что сборка завершена, задает пути разделяемой библиотеки, а затем вызывает исполняемый файл, используя предоставленный шаблон командной строки, вставляя имя программы для заполнителя *%s*.

Другим, особенно полезным примером, является самостоятельный запуск набора тестов. Предположим, что существует тестовый набор *mytest* (его нет). Выше мы использовали скрипт *./test.py* для запуска целого ряда тестов параллельно, неоднократно вызывая реальную тестовую программу, *test-runner*. Вызов *test-runner* непосредственно для одного теста:

```
$ ./waf --run test-runner --command-template="%s --suite=mytest --verbose"
```

Этот аргумент передается *test-runner*. Поскольку *mytest* не существует, будет выдано сообщение об ошибке. Чтобы вывести доступные параметры *test-runner*:

```
$ ./waf --run test-runner --command-template="%s --help"
```

4.3. Отладка

Чтобы запустить программы ns-3 под управлением другой утилиты, например, как отладчик (например, *gdb*) или средство проверки памяти

(например, valgrind), вы используете аналогичную *--command-template = "..."* форму.

Например, чтобы запустить программу ns-3 *hello-simulator* с аргументами <*args*> в отладчике *gdb*:

```
$ ./waf --run=hello-simulator --command-template="gdb %s --args <args>"
```

Обратите внимание, что имя программы ns-3 идет с аргументом *--run*, а утилита управления (здесь gdb) является первой лексемой в аргументе *--commmand-template*. Команда *--args* сообщает *gdb,* что остальная часть командной строки принадлежит программе «низшего уровня». (Некоторые gdb не понимают функцию -args. В этом случае опустите аргументы программы из *--command-template* и используйте команду *gdb set args*.)

Мы можем запустить тест под отладчиком:

```
$ ./waf --run test-runner --command-template="gdb% s --args --suite=mytest --verbose"
```

4.4. Рабочий каталог

Waf должен запускаться из своего расположения в верхней части дерева ns-3. Это станет рабочим каталогом, где будут записываться выходные файлы. Если вы хотите сохранить эти пути в исходном дереве ns-3? Используйте аргумент *--cwd*:

```
$ ./waf --cwd = ...
```

Удобнее начать с вашего рабочего каталога, где вы хотите получить выходные файлы, и в этом случае может помочь:

```
$ function waff {
CWD="$PWD"
cd $NS3DIR>/dev/null
./waf --cwd="$CWD"$*
cd ->/dev/null
}
```

Это сохраняет текущий рабочий каталог, затем инструктируя, Waf меняет рабочий каталог обратно на сохраненный текущий рабочий каталог перед запуском программы.

5. ОСНОВНЫЕ АБСТРАКЦИИ

- Узел
- Приложение
- Канал
- Сетевое устройство
- Пакет

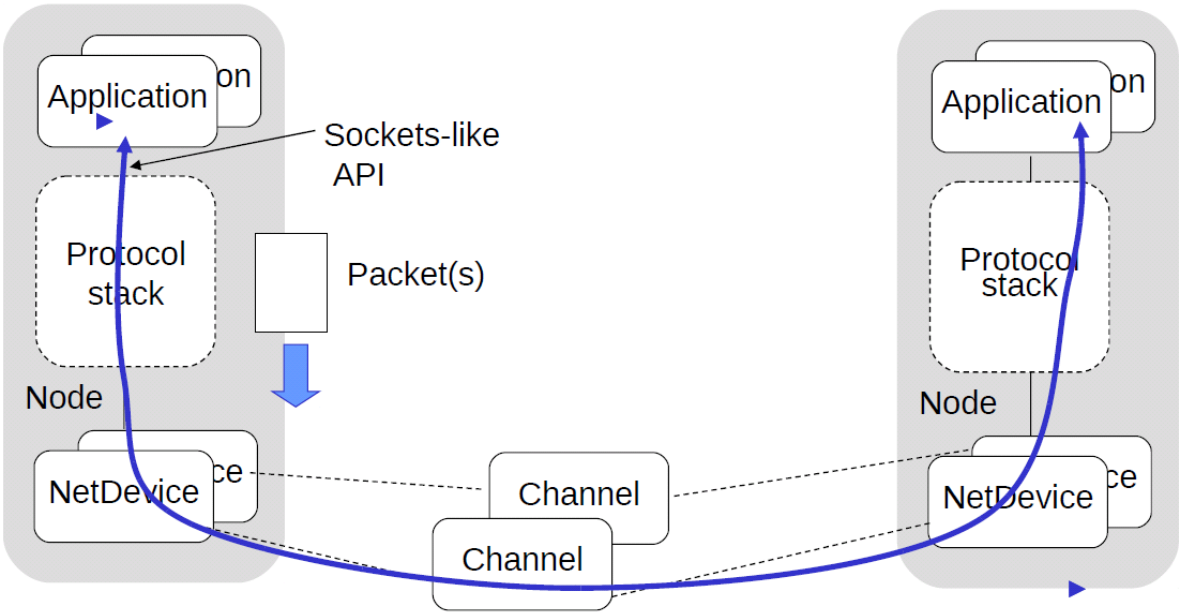

Рис.1. Основные абстракции

Topology Helpers – совокупность функционала модулей для упрощения выполнения общих операций низкоуровневых API

Состоит из:

- container objects(контейнерных объектов)
- Вспомогательных классов

5.1. Создание объектов

Создание объекта происходит с помощью класса Ptr<T >. Например нужно создать узел, имя которого в коде будет «n0»,это будет выглядеть так:

```
Ptr<Node> n0 = CreateObject<Node> ();
```

6. СОЗДАНИЕ СКРИПТА

При написании первого для вас скрипта для NS-3, вы практически выполните первую лабораторную работу. Ознакомьтесь с заданием:

Первая строка в файле - это строка режима emacs. Это сообщает emacs о соглашениях, о форматировании (стиль кодирования), которые мы используем в нашем исходном коде.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
```

Строка режима emacs выше упрощает получение форматирования, если вы используете редактор emacs.

Симулятор ns-3 лицензируется с использованием GNU General Public License. Вы увидите соответствующее GNU legalese во главе каждого файла в дистрибутиве ns-3.

```
// GPLv2 Licence …
```

6.1. Модули Includes

Код собственно начинается с ряда операторов *include*.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"
```

Чтобы помочь нашим высокоуровневым скриптовым пользователям иметь дело с большим количеством включенных файлов, присутствующих в системе, мы группируем их в соответствии с относительно большими модулями. Мы предоставляем один файл *include*, который рекурсивно загружает все включенные файлы, используемые в каждом модуле. Вместо того чтобы искать именно тот заголовок, который вам нужен, и, возможно, получение нужного правильного количества зависимостей, у вас есть возможность загружать группу файлов с большой степенью детализации. Это не самый эффективный подход, но он значительно упрощает написание сценариев.

Каждый из файлов ns-3 *include* помещается в каталог с именем ns3 (в каталоге сборки) во время процесса сборки, чтобы избежать конфликтов имен файлов. Файл *ns3/core-module.h* соответствует модулю ns-3, который вы найдете в каталоге *src/core* в вашем загруженном дистрибутиве. Если вы выберете этот каталог, вы найдете большое количество заголовочных файлов. Когда вы выполняете сборку, Waf помещает публичные файлы заголовков в каталог ns3 под соответствующим каталогом *build/debug* или *build/optimized* в зависимости от вашей конфигурации. Waf также автоматически генерирует модуль *include file* для загрузки всех общедоступных файлов заголовков.

6.2. Пространство имен NS-3

Следующая строка в скрипте - это объявление пространства имен.

```
using namespace ns3;
```

Проект ns-3 реализуется в пространстве имен C ++, называемом ns3. Это группирует все связанные с ns-3 объявления в области вне глобального пространства имен, что поможет с интеграцией с другим кодом. Инструкция using C ++ добавляет пространство имен ns-3 в текущую (глобальную) декларативную область.

6.3. Вход в систему

Следующая строка сценария выглядит следующим образом:

```
NS_LOG_COMPONENT_DEFINE («FirstScriptExample»);
```

Эта строка объявляет компонент ведения журнала, называемый FirstScriptExample, который позволяет включать и отключать ведение журнала сообщений консоли со ссылкой на имя.

6.4. Функция Main

Следующие строки скрипта:

```
 int
 main (int argc, char  *argv [ ])
{
```

Это просто объявление основной функции вашей программы (скрипта). Как и в любой программе на C ++, вам нужно определить основную функцию, которая будет первым запуском функции. Здесь нет ничего особенного. Ваш сценарий ns-3 - это всего лишь программа на C ++.

Следующая строка устанавливает временное разрешение на одну наносекунду, которая по умолчанию является значением:

```
Time :: SetResolution (Time :: NS);
```

Разрешение – это наименьшее значение времени, которое может быть представлено (а также наименьшая представляемая разница между двумя временными значениями). Вы можете изменить разрешение один раз. Механизм, обеспечивающий такую гибкость, несколько перегружает память, поэтому, как только разрешение будет установлено явно, мы освободим память, предотвращая дальнейшие обновления.

Следующие две строки сценария используются для включения двух компонентов протоколирования, встроенных в *EchoClient* и *EchoServerApplication*:

```
LogComponentEnable («UdpEchoClientApplication», LOG_LEVEL_INFO);
LogComponentEnable («UdpEchoServerApplication», LOG_LEVEL_INFO);
```

Если вы прочитали документацию компонента Logging, вы увидите, что существует множество уровней ведомости / детализации журналирования, которые вы можете включить для каждого компонента. Эти две строки кода позволяют вести журнал отладки на уровне INFO для эхо-клиентов и серверов.

Это приведет к тому, что приложение выведет сообщения, как пакеты будут отправляться и приниматься во время моделирования.

6.5. Объекты топологии

Node

В интернет-жаргоне вычислительное устройство, которое подключается к сети, называется host. Поскольку ns-3 является сетевым симулятором, а не симулятором Интернета, не используется термин host,

поскольку он тесно связан с Интернетом и его протоколами. Вместо этого используется более общий термин, также используемый другими симуляторами, который возникает из теории графов – узел.

В ns-3 абстракция базового вычислительного устройства называется узлом. Эта абстракция представлена в C ++ классом Node. Класс Node предоставляет методы для управления представлениями вычислительных устройств в симуляторах.

Следующие строки кода в нашем скрипте будут фактически создавать узлы ns-3, используемые в топологии, которые будут представлять компьютеры в симуляции.

```
Ptr<Node> n0 = CreateObject<Node> ();
Ptr<Node> n1 = CreateObject<Node> ();
Ptr<Node> n2 = CreateObject<Node> ();
…
```

Затем нужно будет создать контейнер для всех узлов

Вспомогательный объект топологии NodeContainer предоставляет удобный способ создания, управления и доступа к любым объектам Node, которые мы создаем для запуска моделирования.

```
NodeContainer c = NodeContainer (n0, n1, n2);
```

Так же после этого нам необходимо будет создать контейнеры узлов для каждой связи, это делается так:

```
NodeContainer c1 = NodeContainer(n0, n1);
NodeContainer c2 = NodeContainer(n1, n2);
```

Если не обходимо будет создание контейнера, который содержит некоторое количество узлов, то это должно будет выглядеть так:

```
NodeContainer c;
c.Create(2);
```

Первая строка выше просто объявляет NodeContainer c, который называется узлом. Вторая строка вызывает метод Create для объекта c, и создаются два узла.

Следующим шагом в построении топологии является объединение наших узлов в сеть. Простейшим видом сети, которую мы поддерживаем, является единственная точка-точка связи между двумя узлами.

PointToPointHelper

Напомним, что двумя из наших ключевых абстракций являются NetDevice и Channel. В реальном мире эти условия примерно соответствуют периферийным картам и сетевым кабелям. Наши вспомогательные объекты по топологии следуют этой тесной связи, и поэтому вы будете использовать единственный PointToPointHelper для настройки и подключения объектов ns-3 PointToPointNetDevice и PointToPointChannel в этом скрипте.

Создаем канал, который используется для связи между узлами.

Первая строка,

```
PointToPointHelper p2p;
```

создает экземпляр объекта PointToPointHelper в стеке. Для этого используем соответствующий класс, с помощью которого указываем параметры канала.

Следующая строка,

```
p2p.SetDeviceAttribute ("DataRate", StringValue ("2Mbps"));
```

сообщает объекту PointToPointHelper, что при создании объекта устройства в качестве скорости передачи данных задается значение «2Мбит/с».

Так же на официальном сайте вы сможете найти полный список атрибутов, определенных для устройств.

Подобно «DataRate» в PointToPointNetDevice вы найдете атрибут «Delay», связанный с PointToPointChannel.

```
p2p.SetChannelAttribute ("Delay", StringValue ("1ms"));
```

Сообщает PointToPointHelper использовать значение «1ms» в качестве значения задержки распространения каждого канала точка-точка, который он впоследствии создает.

Поддержка создания очереди для топологии «точка-точка» задается следующей строкой:

```
p2p.SetQueue ("ns3::DropTailQueue");
```

NetDeviceContainer

Далее устанавливаем устройства и каналы, которые используются в нашей топологии.

На этом этапе сценария у нас есть NodeContainer, который содержит некоторое количество узлов. У нас есть PointToPointHelper, который готов для создания PointToPointChannel между объектами PointToPointNetDevices. Точно так же, как мы использовали вспомогательный объект топологии NodeContainer для создания узлов для нашего моделирования, мы попросим PointToPointHelper выполнить работу, связанную с созданием, настройкой и установкой наших устройств для нас. Нам понадобится список всех создаваемых NetDevice-объектов, поэтому мы используем NetDeviceContainer для их хранения так же, как мы использовали NodeContainer для хранения узлов, которые мы создали. Следующая строка скрипта:

```
NetDeviceContainer devices1 = p2p.Install (c1);
```

…

Так же в настройках мы можем указать очередь, которая будет использоваться, указать скорость передачи данных и задать значение задержки канала.

И не забываем, создать контейнер для всех устройств:

```
NetDeviceContainer d = NetDeviceContainer ();
d.Add(devices1);
…
```

InternetStackHelper

Теперь у нас настроены узлы и устройства, и нам нужно установить сетевые стеки на узлах.

Следующие две строки кода позаботятся об этом.

```
InternetStackHelper internet;
internet.Install (c);
```

InternetStackHelper - это вспомогательный объект топологии, который связывает с Интернетом то, что PointToPointHelper предназначен для двухточечных сетевых устройств. Метод Install принимает в качестве параметра NodeContainer. Когда он будет выполнен - установит Internet Stack (TCP, UDP, IP и т. д.) на каждом узле в контейнере узла.

Ipv4AddressHelper

Далее нам нужно связать устройства на наших узлах с IP-адресами. Мы предоставляем вспомогательный объект топологии для управления распределением IP-адресов. Единственный видимый пользователем интерфейс API - это настройка базового IP-адреса и маски сети, используемых при выполнении фактического распределения адресов (что выполняется на более низком уровне внутри помощника).

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
address.Assign(devices1);
…
```

Тем самым мы сообщаем объекту Ipv4AddressHelper, что он должен начать распределять IP-адреса из сети 10.1.1.0, используя маску 255.255.255.0.

Следующая строка кода:

```
Ipv4InterfaceContainer interfaces = address.Assign (devices2);
```

С помощью Ipv4InterfaceContainer мы будем обращаться к адресу устройств.

Поскольку мы на самом деле построили здесь межсетевую связь, нам нужна некоторая форма межсетевой маршрутизации. Настройка этой формы маршрутизации:

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Теперь у нас есть построенная точка-точка сеть, с установленными стеками и назначенными IP-адресами. На этом этапе нам нужны приложения для генерации трафика.

6.6. Приложения

Еще одна из основных абстракций системы ns-3 - это приложение. В этом скрипте мы используем две специализации приложения core ns-3 класса Application, называемых UdpEchoServerApplication и UdpEchoClientApplication. Как и в наших предыдущих объяснениях, мы используем вспомогательные объекты, чтобы помочь сконфигурировать и управлять базовыми объектами. Здесь мы используем объекты

UdpEchoServerHelper и UdpEchoClientHelper, чтобы сделать нашу жизнь проще.

Первый шаг - это создание и настройка приложения, которое будет принимать пакеты.

Если помощнику конструктора не предоставлен номер порта, который будет принимать пакеты, мы не сможем разместить необходимые атрибуты. Поэтому:

```
uint16_t port = 50000;
```

Следующие две строки:

```
Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (),port));
PacketSinkHelper sinkHelper ("ns3::UdpSocketFactory", sinkLocalAddress);
```

Этот код создает PacketSinkHelper и сообщает ему о создании сокетов с помощью класса ns3::UdpSocketFactory

```
ApplicationContainer sinkApp = sinkHelper.Install (n2);
```

Теперь мы видим, что sinkHelper.Install собирается установить sinkApp на узел, найденный в индексе номер один из NodeContainer, который мы использовали для управления нашими узлами.

Приложению требуется время, чтобы «начать» генерировать трафик, и может потребоваться дополнительное время для «остановки».

Эти времена устанавливаются с помощью методов ApplicationContainer Start и Stop.

```
sinkApp.Start (Seconds (1.0));
sinkApp.Stop (Seconds (10.0));
```

Второй шаг - создаем приложение, которое будет генерировать и отправлять пакеты на сервер.

```
OnOffHelper clientHelper ("ns3::UdpSocketFactory", Address ());
clientHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
clientHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
```

Третий шаг - создаем клиента при помощи clientHelper, который используется для связи с сервером

Приложение-клиент настраивается способом, по существу аналогичным методу для приложения, которое будет принимать пакеты.

```
ApplicationContainer clientApps;
AddressValue remoteAddress (InetSocketAddress (interfaces.GetAddress(1),
port));
```

Мы передаем параметры, которые используются (внутренне для помощника), чтобы установить атрибуты «RemoteAddress» и «RemotePort» в соответствии с нашим соглашением для создания необходимых параметров Attributes в хелперных конструкторах.

```
clientHelper.SetAttribute ("Remote", remoteAddress);
```

Устанавливаем узел, который у нас будет в роли клиента.

```
clientApps.Add (clientHelper.Install (n0));
```

Клиенту можем указать атрибут «MaxPackets», что задает максимальное количество пакетов, которые мы разрешаем отправлять во

время симуляции. Атрибут «Interval» указывает клиенту, сколько времени ждать между пакетами, а атрибут «PacketSize» указывает клиенту размер принимаемого пакета.

```
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

Как и в случае с получателя пакетов, мы указываем клиенту Start и Stop, но здесь мы запустим клиента через 2 секунду после включения сервера и приостановим через 10 секунд после симуляции.

6.7. Анимация

Переходим к завершающей части нашего кода, создаем файл для анимации.

```
AnimationInterface anim("animLab1.xml");
```

Затем устанавливаем позиции (X и Y) для каждого узла.

```
anim.SetConstantPosition(n0, 5.0, 10.0);
anim.SetConstantPosition(n1, 10.0, 10.0);
…
```

После симуляции открываем в NetAnim наш файл "animLab1.xml".

6.8. Трассировка

Затем нам нужно использование системы трассировки. Трассировочная система ns-3 построена на концепциях независимых источников трассировки и трассирующих приемников и единого механизма для подключения источников к приемникам. Источники трассировки - это объекты, которые могут сигнализировать о событиях, происходящих в симуляции, и предоставлять доступ к интересным базовым данным. Например, источник трассировки мог указать, когда пакет получен сетевым устройством, и предоставить доступ к содержимому пакета заинтересованным приемникам трассировки. Трассирующие приемники являются потребителями событий и данных, предоставляемых источниками трассировки. Например, можно создать приемник трассировки, который (при подключении к источнику трассировки предыдущего примера) выводит интересные части полученного пакета.

Добавим некоторый вывод трассировки ASCII к нашему скрипту

```
AsciiTraceHelper ascii;
p2p.EnableAsciiAll (ascii.CreateFileStream ("lab1.tr"));
```

Этот код использует вспомогательный объект, чтобы помочь создавать трассировки ASCII.

Вторая строка содержит два вложенных вызова метода. Метод «inside», CreateFileStream (), чтобы создать объект потока файлов в стеке (без имени объекта) и передать его вызываемому методу.

Внешний вызов EnableAsciiAll () сообщает помощнику, что вы хотите включить трассировку ASCII на всех устройствах точка-точка в вашем

моделировании; а так же предоставленные трассировки записывали информацию о перемещении пакетов в формате ASCII. Файл открывается с помощью TraceMetrics.

6.9. Запуск моделирования

Симуляция остановится автоматически, когда в очереди событий не появятся другие события или когда будет найдено специальное событие Стоп. Событие *Stop* создается через функцию *Simulator :: Stop (stopTime);*

Simulator :: Stop необходимо для грамотной остановки моделирования.

```
Simulator::Stop( Seconds(10.0));
```

Тем самым моделирование естественным образом заканчивается через 10 секунд.

Важно вызвать Simulator :: Stop перед вызовом Simulator :: Run;. В противном случае, Simulator :: Run никогда не сможет вернуть управление основной программе для выполнения остановки!

```
Simulator :: Run ();
Simulator::Destroy ();
return 0;
}
```

7. СОБРАННЫЙ КОД

Изучив задание, что нам нужно построить и промоделировать, и подробное описание выполнения первой лабораторной работы должен получится код(см.ниже):

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"

using namespace ns3;

int
main (int argc, char *argv[])
{
 Time::SetResolution (Time::NS);

  Ptr<Node> n0 = CreateObject<Node> ();
  Ptr<Node> n1 = CreateObject<Node> ();
  Ptr<Node> n2 = CreateObject<Node> ();
  Ptr<Node> n3 = CreateObject<Node> ();

  NodeContainer c = NodeContainer (n0, n1, n2, n3);

  NodeContainer c1 = NodeContainer(n0, n1);
  NodeContainer c2 = NodeContainer(n1, n2);
  NodeContainer c3 = NodeContainer(n1, n3);
```

Подключаемые модули, которые будут задействованы

ns-3 project namespace

```
     PointToPointHelper p2p;
     p2p.SetQueue ("ns3::DropTailQueue");
     p2p.SetDeviceAttribute ("DataRate", StringValue ("2Mbps"));
     p2p.SetChannelAttribute ("Delay", StringValue ("1ms"));

     NetDeviceContainer devices1 = p2p.Install (c1);

     p2p.SetQueue ("ns3::RedQueue");
     p2p.SetDeviceAttribute ("DataRate", StringValue ("512kbps"));
     p2p.SetChannelAttribute ("Delay", StringValue ("10ms"));

     NetDeviceContainer devices2 = p2p.Install (c2);

     p2p.SetQueue ("ns3::DropTailQueue");
     p2p.SetDeviceAttribute ("DataRate", StringValue ("64kbps"));
     p2p.SetChannelAttribute ("Delay", StringValue ("100ms"));

     NetDeviceContainer devices3 = p2p.Install (c3);

     NetDeviceContainer d = NetDeviceContainer ();
     d.Add(devices1);
     d.Add(devices2);
     d.Add(devices3);


     InternetStackHelper internet;
     internet.Install (c);

     Ipv4AddressHelper address;
     address.SetBase ("10.1.1.0", "255.255.255.0");
     address.Assign(devices1);
     address.SetBase ("10.1.2.0", "255.255.255.0");
     address.Assign(devices2);
     address.SetBase ("10.1.3.0", "255.255.255.0");
     Ipv4InterfaceContainer interfaces = address.Assign(devices3);

     Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

    uint16_t port = 50000;
     Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (),
port));
     PacketSinkHelper sinkHelper ("ns3::UdpSocketFactory",
sinkLocalAddress);
     ApplicationContainer sinkApp = sinkHelper.Install (n3);
     sinkApp.Start (Seconds (1.0));
     sinkApp.Stop (Seconds (10.0));

     OnOffHelper clientHelper ("ns3::UdpSocketFactory", Address ());
     clientHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
     clientHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));

     ApplicationContainer clientApps;
     AddressValue remoteAddress (InetSocketAddress
(interfaces.GetAddress(1), port));
     clientHelper.SetAttribute ("Remote", remoteAddress);
     clientApps.Add (clientHelper.Install (n0));
     clientApps.Start (Seconds (2.0));
     clientApps.Stop (Seconds (10.0));
```

Конфигурация топологии

Подключение интернет стека

Подключение приложен

```
  AnimationInterface anim("animLab1.xml");
  anim.SetConstantPosition(n0, 5.0, 10.0);
  anim.SetConstantPosition(n1, 10.0, 10.0);
  anim.SetConstantPosition(n2, 15.0, 5.0);
  anim.SetConstantPosition(n3, 15.0, 15.0);
```

Файл для анимации

```
  AsciiTraceHelper ascii;
  p2p.EnableAsciiAll (ascii.CreateFileStream ("lab1.tr"));
```

Трассировка

```
  Simulator::Stop( Seconds(10.0));
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Запуск моделирования

7.1. Запуск скрипта

У нас есть готовый скрипт, который мы можем запустить. Теперь все, что вам нужно сделать,- это сбросить свой скрипт в каталог scratch, и он автоматически будет создан, если вы запустите Waf.

Теперь запустите свой первый пример сценария, используя waf:

```
$ ./waf
```

Вы должны увидеть сообщения, сообщающие, что ваш пример lab1 был успешно создан.

```
Waf: Entering directory `/home/student/ns-allinone-3.20/ns-3.20/build'
[1970/2306] cxx: scratch/lab1.cc -> build/scratch/lab1.cc.1.o
[2284/2306] cxxprogram: build/scratch/lab1.cc.1.o -> build/scratch/lab1
Waf: Leaving directory `/home/student/ns-allinone-3.20/ns-3.20/build'
'build' finished successfully (28.883s)
```

Теперь вы можете запустить пример (обратите внимание, что если вы создаете свою программу в каталоге scratch, вы должны запустить ее из каталога scratch):

```
$ ./waf --run scratch/lab1
```

8. ПОСТРОЕНИЕ СЕТИ С ТОПОЛОГИЕЙ «ШИНА»

В этом разделе мы собираемся расширить наше мастерство сетевых устройств и каналов ns-3, охватив пример сети шины. Ns-3 обеспечивает сетевое устройство и канал, который мы вызываем CSMA (множественный доступ с поддержкой несущей). Устройство ns-3 CSMA моделирует простую сеть Ethernet. Реальный Ethernet использует схему CSMA / CD (множественный доступ с контролем несущей с обнаружением столкновений) с экспоненциально увеличивающейся задержкой для борьбы за общую среду передачи. Устройство и канал ns-3 CSMA моделирует только подмножество. Точно так же, как мы видели вспомогательные объекты топологии «точка-точка» при построении топологий «точка-точка», в этом разделе мы увидим эквивалентные помощники топологии CSMA. Появление и работа этих помощников должны быть вам знакомы. Предоставляется пример сценария. Этот скрипт основывается на скрипте,

который вы создавали до этого и добавляется сеть CSMA к моделированию «точка-точка», которое мы уже рассматривали. Как и в примере, который рассматривался выше, файл начинается с строки режима emacs и некоторого шаблона GPL.

Одной вещью, которая может быть полезной, является небольшая часть ASCII-искусства, которая показывает мультипликацию топологии сети, построенной в примере. В этом случае вы можете увидеть, что мы собираемся расширить наш пример «точка-точка» (связь между узлами n0 и n1 ниже), повесив сеть шины с правой стороны. Обратите внимание, что это топология сети по умолчанию, так как фактически вы можете изменять количество узлов, созданных в локальной сети. Если вы установите nCsma в один, в локальной сети будет два узла (канал CSMA) - один требуемый узел и один «лишний» узел. По умолчанию есть три дополнительных узла, как показано ниже:

```
// Default Network Topology
//
//       10.1.1.0
// n0 -------------- n1   n2   n3   n4
//    point-to-point |    |    |    |
//                   ================
//                     LAN 10.1.2.0
```

Затем используется пространство имен ns-3 и определяется компонент ведения журнала.

```
using namespace ns3; NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");
```

Основная часть кода начинается немного по-другому. Мы используем многословный флаг, чтобы определить, включены ли компоненты ведения журнала UdpEchoClientApplication и UdpEchoServerApplication. По умолчанию этот флажок равен true (компоненты ведения журнала включены), но он позволяет нам отключить ведение журнала во время регрессивного тестирования этого примера.

Вы увидите знакомый код, который позволит вам изменить количество устройств в сети CSMA с помощью аргумента командной строки Последняя строка гарантирует, что у вас есть хотя бы один «лишний» узел.

```
bool verbose = true;
uint32_t nCsma = 3;
CommandLine cmd;
cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
cmd.Parse (argc, argv);
if (verbose)
{
LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
}
nCsma = nCsma == 0 ? 1 : nCsma;
```

Следующий шаг - создать два узла, которые будут соединяться по связи «точка-точка». NodeContainer используется для этого, как это было сделано в первом вашем скрипте:

```
NodeContainer p2pNodes;
```

```
p2pNodes.Create (2);
```

Затем мы объявляем еще один NodeContainer, чтобы удерживать узлы, которые будут частью сети шины (CSMA). Первая строка, мы просто создаем экземпляр самого объекта-контейнера.

```
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
```

Следующие строки кода получают первый узел (как имеющий индекс один) из контейнера узла «точка-точка» и добавляет его в контейнер узлов, которые будут получать устройства CSMA. Узел, о котором идет речь, в конечном итоге будет иметь устройство «точка-точка» и устройство CSMA. Затем мы создаем несколько «лишних» узлов, которые составляют оставшуюся часть сети CSMA. Поскольку у нас уже есть один узел в сети CSMA - тот, который будет иметь как устройство «точка-точка», так и сетевое устройство CSMA, количество «лишних» узлов означает количество нужных вам узлов в секции CSMA минус один.

Следующие строки кода должны быть уже хорошо знакомы. Мы создаем экземпляр PointToPointHelper и устанавливаем связанные атрибуты по умолчанию, чтобы мы создавали передатчик на пять мегабит в секунду на устройствах, созданных с помощью помощника, и задержку в две миллисекунды на каналах, созданных помощником.

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

Затем мы создаем экземпляр NetDeviceContainer для отслеживания сетевых устройств точка-точка и устанавливаем устройства точка-точка на узлах.

CsmaHelper работает точно так же, как PointToPointHelper, но он создает и соединяет устройства и каналы CSMA. В случае устройства CSMA и пары каналов обратите внимание на то, что скорость передачи данных определяется атрибутом канала вместо атрибута устройства. Это связано с тем, что реальная сеть CSMA не позволяет смешивать, например, 10Base-T и 100Base-T устройства в данном канале. Сначала мы установили скорость передачи данных до 100 мегабит в секунду, а затем установили задержку скорости света канала на 6560 наносекунд. Обратите внимание, что вы можете установить атрибут, используя его собственный тип данных.

```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

Так же, как мы создали NetDeviceContainer для хранения устройств, созданных PointToPointHelper, мы создаем NetDeviceContainer для хранения устройств, созданных нашим CsmaHelper. Мы вызываем метод

Install CsmaHelper для установки устройств в узлы csmaNodes NodeContainer.

Теперь у нас есть наши узлы, устройства и каналы, но у нас нет стеков протоколов. Мы будем использовать InternetStackHelper для установки этих стеков.

```
InternetStackHelper stack;
stack.Install (p2pNodes.Get (0));
stack.Install (csmaNodes);
```

Мы будем использовать Ipv4AddressHelper для назначения IP-адресов нашим интерфейсам устройств. Сначала мы используем сеть 10.1.1.0 для создания двух адресов, необходимых для двух наших устройств точка-точка.

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);
```

Мы сохраняем созданные интерфейсы в контейнере, чтобы упростить вынос информации адресации позже для использования при настройке приложений.

Теперь нам нужно назначить IP-адреса нашим интерфейсам устройств CSMA. Операция работает так же, как и для случая «точка-точка», за исключением того, что мы теперь выполняем операцию над контейнером, который имеет переменное количество CSMA-устройств, - помните, что мы сделали количество CSMA-устройств изменяемым аргументом командной строки. В этом случае устройства CSMA будут связаны с IP-адресами из номера сети 10.1.2.0.

```
address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);
```

Теперь у нас есть топология, но нам нужны приложения. Мы собираемся создать экземпляр сервера на одном из узлов, у которого есть устройство CSMA, а клиент на узле имеет только устройство точка-точка.

Мы создаем UdpEchoServerHelper и предоставляем требуемое значение атрибута конструктору, который является номером порта сервера. Этот порт может быть изменен позже, если это необходимо, с помощью метода SetAttribute, но мы требуем, чтобы он был предоставлен конструктору.

```
UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get
(nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

Напомним, что узел NodeContainer csmaNodes содержит один из узлов, созданных для сети точка-точка, и nCsma «лишних» узлов. Нулевой вход контейнера csmaNodes будет узлом точка-точка. Мы создадим один «дополнительный» узел CSMA, тогда он будет в индексе один из контейнера csmaNodes. Если мы создадим nCsma «лишние» узлы,

последний будет в индексе nCsma. Вы видите, что это выставляется в Get первой строки кода.

Клиентское приложение настриваем точно так же, как мы делали раньше. Опять же, мы предоставляем необходимые атрибуты для UdpEchoClientHelper в конструкторе (в данном случае это удаленный адрес и порт). Мы говорим клиенту отправлять пакеты на сервер, который мы только что установили на последнем из «лишних» узлов CSMA. Мы устанавливаем клиент на самый левый узел «точка-точка», показанный на иллюстрации топологии.

```
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

Поскольку мы на самом деле построили здесь межсетевую связь, нам нужна некоторая форма межсетевой маршрутизации. Настройка этой формы маршрутизации:

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Затем мы включаем трассировку pcap. Первая строка кода - должна включать трассировку pcap в PointToPointHelper, должна быть вам знакома. Вторая строка включает трассировку pcap в CsmaHelper.

```
pointToPoint.EnablePcapAll ("second");
csma.EnablePcap ("second", csmaDevices.Get (1), true);
```

Сеть CSMA представляет собой многоточечную сеть. Это означает, что на общем носителе могут несколько конечных точек. У каждой из этих конечных точек есть сетевое устройство, связанное с ним. Существуют две основные альтернативы для сбора информации трассировки из такой сети. Одним из способов является создание файла трассировки для каждого сетевого устройства и сохранение только пакетов, которые передаются или принимаются этим сетевым устройством. Другой способ - выбрать одно из устройств и поместить его в беспорядочный режим. Это единственное устройство затем «обнюхивает» сеть для всех пакетов и сохраняет их в одном файле pcap. Так работает, например, tcpdump. Этот конечный параметр указывает помощнику CSMA, нужно или нет организовывать захват пакетов в беспорядочном режиме.

В этом примере мы выберем одно из устройств в сети CSMA и попросим его выполнить беспорядочное «обнюхивание» сети, тем самым эмулируя, что бы сделал tcpdump. Если вы были на машине с Linux, вы могли бы сделать что-нибудь вроде tcpdump -i eth0, чтобы получить след. В этом случае мы указываем устройство с помощью csmaDevices.Get (1), которое выбирает первое устройство в контейнере. Установка окончательного параметра в true дает возможность проведения случайных захватов.

Последний раздел кода просто воспроизводит запуск и остановку симуляции.

```
Simulator::Run ();
Simulator::Destroy ();
return 0;
}
```

9. ПОСТРОЕНИЕ БЕСПРОВОДНОЙ СЕТЕВОЙ ТОПОЛОГИИ

В этом разделе мы собираемся освоить пример беспроводной сети. Ns-3 предоставляет набор моделей 802.11, которые пытаются обеспечить точную реализацию MAC-уровня спецификации 802.11 и модель PHY - уровня стандарта 802.11a.

Точно так же, как мы видели, как вспомогательные объекты топологии «точка-точка» и CSMA при построении топологий «точка-точка», в этом разделе мы увидим эквивалентные помощники топологии Wi-Fi.

Этот скрипт основывается на скрипте построения сети с топологией шина и добавляет сеть Wi-Fi.

Как и во всех скриптах ns-3, файл начинается со строки режима emacs и некоторого шаблона GPL.

Взгляните на искусство ASCII (воспроизводится ниже), которое показывает стандартную топологию сети, построенную в этом примере. Вы можете видеть, что мы собираемся расширить наш пример, повесив беспроводную сеть. Обратите внимание, что это топология сети по умолчанию, так как вы можете фактически изменять количество узлов, созданных в проводных и беспроводных сетях. Как и в предыдущем сценарии, если вы измените nCsma, он даст вам несколько «лишних» узлов CSMA. Аналогичным образом, вы можете установить nWifi для управления количеством узлов STA (станции), созданных в симуляции. В беспроводной сети всегда будет один узел точки доступа (точки доступа). По умолчанию есть три «лишних» узла CSMA и три беспроводных узла STA.

Код начинается с загрузки файлов включения модулей. В комплект поставки входит несколько новых модулей, соответствующих модулю Wifi и модулю мобильности.

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
```

Иллюстрация топологии сети выглядит следующим образом:

```
// Default Network Topology
//
// Wifi 10.1.3.0
```

```
//                AP
// *     *     *     *
// |     |     |     |       10.1.1.0
// n5    n6    n7    n0 -------------- n1   n2   n3   n4
//                    point-to-point |    |    |    |
//                                   ================
//                                     LAN 10.1.2.0
```

Вы видите, что мы добавляем новое сетевое устройство к узлу слева от линии связи точка-точка, которая становится точкой доступа для беспроводной сети. Для создания новой сети 10.1.3.0 создается несколько беспроводных узлов STA.

После иллюстрации используется пространство имен ns-3 и определяется компонент ведения журнала.

```
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("ThirdScriptExample");
```

Основной код начинается так же, предыдущем сценарии, добавляя некоторые параметры командной строки для включения или отключения регистрации компонентов и изменения количества созданных устройств.

```
bool verbose = true;
uint32_t nCsma = 3;
uint32_t nWifi = 3;
CommandLine cmd;
cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

cmd.Parse (argc,argv);

if (verbose)
  {
   LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
   LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
  }
```

Как и во всех предыдущих примерах, следующим шагом будет создание двух узлов, которые мы будем подключать по связи точка-точка.

```
NodeContainer p2pNodes;
p2pNodes.Create (2);
```

Создаем экземпляр PointToPointHelper и устанавливаем атрибуты по умолчанию. Затем мы устанавливаем устройства на узлы и канал между ними.

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

Следующей строчкой мы объявляем еще один NodeContainer, для узлов, которые будут частью сети шины (CSMA).

```
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
```

Следующая строка кода получает первый узел (как имеющий индекс один) из контейнера узла «точка-точка» и добавляет его в контейнер узлов,

которые будут получать устройства CSMA. Узел, о котором идет речь, в конечном итоге будет оснащен устройством точка-точка и устройством CSMA. Затем мы создаем несколько «лишних» узлов, которые составляют оставшуюся часть сети CSMA.

Затем мы создаем экземпляр CsmaHelper и устанавливаем его атрибуты, как в предыдущем примере. Мы создаем NetDeviceContainer для отслеживания созданных сетевых устройств CSMA, а затем устанавливаем CSMA-устройства на выбранных узлах.

```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

Далее мы создадим узлы, которые будут частью сети Wifi. Мы собираемся создать несколько узлов «станции», как указано в аргументе командной строки, и мы собираемся использовать «самый левый» узел связи точка-точка в качестве узла для точки доступа.

```
NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);
```

Следующий строки кода формируют wifi-устройства и канал межсоединений между этими wifi узлами. Сначала мы настроим помощники PHY и канала:

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
```

Для простоты этот код использует конфигурацию PHY- уровня по умолчанию и модели каналов, которые документированы в документации API doxygen для методов *YansWifiChannelHelper :: Default* и *YansWifiPhyHelper :: Default.* После создания этих объектов мы создаем объект канала и связываем его с нашим диспетчером объектов PHY-уровня, чтобы убедиться, что все объекты уровня PHY, созданные YansWifiPhyHelper, используют один и тот же базовый канал.

```
phy.SetChannel (channel.Create ());
```

Когда настроен помощник PHY, мы можем сосредоточиться на уровне MAC. Здесь мы выбираем работать с non-Qos MAC, поэтому мы используем объект NqosWifiMacHelper для установки MAC-параметров.

```
WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
```

Метод SetRemoteStationManager сообщает помощнику тип используемого алгоритма управления скоростью. Здесь он просит помощника использовать алгоритм AARF - подробности, конечно же, доступны в Doxygen.

Далее мы настроим тип MAC, SSID для сети инфраструктуры, которую мы хотим настроить и убедиться, что наши станции не выполняют активного зондирования:

```
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
"Ssid", SsidValue (ssid),
"ActiveProbing", BooleanValue (false));
```

Этот код сначала создает объект идентификатора набора служб 802.11 (SSID), который будет использоваться для установки значения атрибута Ssid для реализации уровня MAC. Особый уровень MAC-уровня, который будет создан помощником, определяется атрибутом как тип *«ns3::StaWifiMac»*. Использование NqosWifiMacHelper гарантирует, что атрибут «QosSupported» для созданных объектов MAC установлен в false. Комбинация этих двух конфигураций означает, что следующий экземпляр MAC будет являться станцией, отличной от QoS (STA), в инфраструктуре BSS (то есть BSS с AP). И, наконец, атрибут «ActiveProbing» установлен в false. Это означает, что запросы зондов не будут отправляться MAC-адресами, созданными этим помощником.

После того, как все параметры, относящиеся к конкретной станции, полностью настроены, как на уровнях MAC, так и PHY, мы можем вызвать наш знакомый метод Install для создания Wi-Fi-устройств этих станций:

```
NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);
```

Мы настроили Wifi для всех наших узлов STA, и теперь нам нужно настроить узел точки доступа. Мы начнем этот процесс, изменив атрибуты по умолчанию для NqosWifiMacHelper, чтобы отразить требования AP.

```
mac.SetType ("ns3::ApWifiMac",
"Ssid", SsidValue (ssid));
```

В этом случае NqosWifiMacHelper собирается создать уровни MAC из «ns3::ApWifiMac», последний из которых указывает, что экземпляр MAC, сконфигурированный как AP, должен быть создан, а вспомогательный тип подразумевает, что должен быть установлен атрибут «QosSupported» False - отключение поддержки QoS в соответствии с 802.11e / WMM в созданных точках доступа.

В следующих строках создается единая точка доступа, которая использует один и тот же набор атрибутов (и канала) уровня PHY в качестве станций:

```
NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);
```

Теперь мы собираемся добавить модели мобильность. Мы хотим, чтобы узлы STA были мобильными, блуждающими внутри ограничительной рамки, и мы хотим сделать узел AP неподвижным. Мы используем MobilityHelper, чтобы сделать это проще для нас. Во-первых, мы создаем экземпляр объекта MobilityHelper и устанавливаем некоторые атрибуты, управляющие функциональностью «позиционный распределитель».

```
MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
"MinX", DoubleValue (0.0),
```

```
"MinY", DoubleValue (0.0),
"DeltaX", DoubleValue (5.0),
"DeltaY", DoubleValue (10.0),
"GridWidth", UintegerValue (3),
"LayoutType", StringValue ("RowFirst"));
```

Этот код говорит помощнику мобильности использовать двумерную сетку, чтобы первоначально разместить узлы STA. Не стесняйтесь исследовать Doxygen для класса *ns3 :: GridPositionAllocator*, чтобы точно узнать, что делается.

Мы разместили наши узлы на начальной сетке, но теперь нам нужно сказать им, как двигаться. Мы выбираем RandomWalk2dMobilityModel, у которого узлы перемещаются в случайном направлении со случайной скоростью вокруг внутри ограничительной рамки.

```
mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
"Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));
```

Теперь мы сообщаем MobilityHelper, чтобы он установил модели мобильность на узлах STA.

```
mobility.Install (wifiStaNodes);
```

Мы хотим, чтобы точка доступа оставалась в фиксированном положении во время симуляции. Мы достигнем этого, устанавливая модель мобильности для этого узла как *ns3::ConstantPositionMobilityModel*:

```
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);
```

Теперь у нас есть созданные нами узлы, устройства и каналы, а также модели мобильности, выбранные для узлов Wifi, но у нас нет стеков протоколов. Так же, как мы делали этого, мы будем использовать InternetStackHelper для установки этих стеков.

```
InternetStackHelper stack;
stack.Install (csmaNodes);
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);
```

Как и во втором примере сценария, мы будем использовать Ipv4AddressHelper для назначения IP-адресов нашим интерфейсам устройств. Сначала мы используем сеть 10.1.1.0 для создания двух адресов, необходимых для двух наших устройств точка-точка. Затем мы используем сеть 10.1.2.0 для назначения адресов в CSMA-сети, а затем присваиваем адреса из сети 10.1.3.0 как устройствам STA, так и точке доступа в беспроводной сети.

```
Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (staDevices);
address.Assign (apDevices);
```

Мы помещаем эхо-сервер в «самый правый» узел на рисунке в начале файла.

```
UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get
(nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

И мы помещаем эхо-клиента на последний созданный узел STA, указывая его на сервер в сети CSMA.

```
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (wifiStaNodes.Get
(nWifi - 1));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

Поскольку мы создали здесь межсетевую связь, нам нужно включить межсетевую маршрутизацию:

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Имитация, которую мы только что создали, никогда не «естественным образом» прекратится. Это связано с тем, что мы попросили беспроводную точку доступа генерировать маяки. Он будет генерировать маяки навсегда, и это приведет к тому, что события симулятора будут планироваться в будущем неограниченно долго, поэтому мы должны сказать симулятору «Стоп», даже если он может иметь запланированные события генерирования маяков. Следующая строка кода сообщает симулятору остановиться:

```
Simulator::Stop (Seconds (10.0));
```

Мы создаем трассировку для охвата всех трех сетей.

```
pointToPoint.EnablePcapAll ("third");
phy.EnablePcap ("third", apDevices.Get (0));
csma.EnablePcap ("third", csmaDevices.Get (0), true);
```

Эти три строки кода будут запускать pcap-трассировку на обоих узлах точка-точка, которые служат нашей основой, запустит трассировку (мониторинг) в сети Wi-Fi и начнет беспорядочную трассировку в сети CSMA. Это позволит нам увидеть весь трафик с минимальным количеством файлов трассировки.

Финальная часть, мы фактически запускаем симуляцию, останавливаем и затем выходим из программы.

```
Simulator::Run ();
Simulator::Destroy ();
return 0;
}
```

10. ПРИМЕРЫ МОДЕЛИРОВАНИЯ СЕТЕЙ IP НА NS-3

10.1. Пример 1

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"

using namespace ns3;

int
main (int argc, char *argv[])
{
 Time::SetResolution (Time::NS);

 // Создание четырех узлов, используемых в топологии
 Ptr<Node> n0 = CreateObject<Node> ();
 Ptr<Node> n1 = CreateObject<Node> ();
 Ptr<Node> n2 = CreateObject<Node> ();
 Ptr<Node> n3 = CreateObject<Node> ();

 // Создание контейнера для всех узлов
 NodeContainer c = NodeContainer (n0, n1, n2, n3);

 // Создание контейнеров узлов для каждой связи
 NodeContainer c1 = NodeContainer(n0, n1);
 NodeContainer c2 = NodeContainer(n1, n2);
 NodeContainer c3 = NodeContainer(n1, n3);

 // Создаем канал, который используется для связи между узлами
 // Для этого используем соответствующий класс, с помощью которого
указываем
 // параметры канала
 // Далее устанавливаем устройства и каналы, которые используются в
нашей топологии
 PointToPointHelper p2p;
 p2p.SetQueue ("ns3::DropTailQueue");
 p2p.SetDeviceAttribute ("DataRate", StringValue ("2Mbps"));
 p2p.SetChannelAttribute ("Delay", StringValue ("1ms"));

 NetDeviceContainer devices1 = p2p.Install (c1);

 p2p.SetQueue ("ns3::RedQueue");
```

```
p2p.SetDeviceAttribute ("DataRate", StringValue ("512kbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("10ms"));

NetDeviceContainer devices2 = p2p.Install (c2);

p2p.SetQueue ("ns3::DropTailQueue");
p2p.SetDeviceAttribute ("DataRate", StringValue ("64kbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("100ms"));

NetDeviceContainer devices3 = p2p.Install (c3);

// Создаем контейнер для всех устройств
NetDeviceContainer d = NetDeviceContainer ();
d.Add(devices1);
d.Add(devices2);
d.Add(devices3);

// Устанавливаем сетевые стеки на узлах
InternetStackHelper internet;
internet.Install (c);

// Далее раздаем IP-адреса всем устройствам
// С помощью Ipv4InterfaceContainer мы будем обращаться к адресу устройств
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
address.Assign(devices1);
address.SetBase ("10.1.2.0", "255.255.255.0");
address.Assign(devices2);
address.SetBase ("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign(devices3);

// Создание узлов маршрутизации, инициализация базы данных маршрутизации,
// настройка таблицы маршрутизации в узлах
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

// Создание и настройка приложения, которое будет принимать пакеты
uint16_t port = 50000;
Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper sinkHelper ("ns3::UdpSocketFactory", sinkLocalAddress);
```

```
    ApplicationContainer sinkApp = sinkHelper.Install (n3);
    sinkApp.Start (Seconds (1.0));
    sinkApp.Stop (Seconds (10.0));

    // Создаем приложение которое будет генерировать и отправлять
пакеты на сервер
    OnOffHelper clientHelper ("ns3::UdpSocketFactory", Address ());
    clientHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
    clientHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));

    // Создаем клиента при помощи clientHelper, который используется для
связи с сервером
    ApplicationContainer clientApps;
    AddressValue remoteAddress (InetSocketAddress
(interfaces.GetAddress(1), port));
    clientHelper.SetAttribute ("Remote", remoteAddress);
    clientApps.Add (clientHelper.Install (n0));
    clientApps.Start (Seconds (2.0));
    clientApps.Stop (Seconds (10.0));

    // Создаем файл для анимации. Затем устанавливаем позиции (X и Y)
для каждого узла.
    AnimationInterface anim("animLab1.xml");
    anim.SetConstantPosition(n0, 5.0, 10.0);
    anim.SetConstantPosition(n1, 10.0, 10.0);
    anim.SetConstantPosition(n2, 15.0, 5.0);
    anim.SetConstantPosition(n3, 15.0, 15.0);

    // Трейс-файл общего вида. Файл открывается с помощью TraceMetrics
    AsciiTraceHelper ascii;
    p2p.EnableAsciiAll (ascii.CreateFileStream ("lab1.tr"));

    Simulator::Stop( Seconds(10.0));
    Simulator::Run ();
    Simulator::Destroy ();
    return 0;
  }
```

10.2. Пример 2

```
#include "ns3/core-module.h"
```

```
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/olsr-helper.h"
#include "ns3/dsdv-helper.h"

using namespace ns3;

int
main (int argc, char *argv[])
{
 Time::SetResolution (Time::NS);

 // Включаем глобальную динамическую маршрутизацию
 Config::SetDefault ("ns3::Ipv4GlobalRouting::RespondToInterfaceEvents",
BooleanValue (true));

 // Создание контейнера, который содержит 10 узлов
 NodeContainer c;
 c.Create(10);

 // Создание контейнеров для каждой связи между узлами
 NodeContainer n0n1 = NodeContainer(c.Get(0), c.Get(1));
 NodeContainer n1n2 = NodeContainer(c.Get(1), c.Get(2));
 NodeContainer n2n3 = NodeContainer(c.Get(2), c.Get(3));
 NodeContainer n3n4 = NodeContainer(c.Get(3), c.Get(4));
 NodeContainer n4n5 = NodeContainer(c.Get(4), c.Get(5));
 NodeContainer n5n6 = NodeContainer(c.Get(5), c.Get(6));
 NodeContainer n6n7 = NodeContainer(c.Get(6), c.Get(7));
 NodeContainer n7n8 = NodeContainer(c.Get(7), c.Get(8));
 NodeContainer n8n9 = NodeContainer(c.Get(8), c.Get(9));
 NodeContainer n9n0 = NodeContainer(c.Get(9), c.Get(0));
 NodeContainer n1n5 = NodeContainer(c.Get(1), c.Get(5));
 NodeContainer n6n9 = NodeContainer(c.Get(6), c.Get(9));

 // Устанавливаем сетевые стеки для каждого узла
 InternetStackHelper internet;
 internet.Install (c);
```

```
// Создаем и настраиваем канал, который будет применен для связи узлов
PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));

// Далее устанавливаем устройства и каналы, которые используются в нашей топологии
NetDeviceContainer d0d1 = p2p.Install (n0n1);
NetDeviceContainer d1d2 = p2p.Install (n1n2);
NetDeviceContainer d2d3 = p2p.Install (n2n3);
NetDeviceContainer d3d4 = p2p.Install (n3n4);
NetDeviceContainer d4d5 = p2p.Install (n4n5);
NetDeviceContainer d5d6 = p2p.Install (n5n6);
NetDeviceContainer d6d7 = p2p.Install (n6n7);
NetDeviceContainer d7d8 = p2p.Install (n7n8);
NetDeviceContainer d8d9 = p2p.Install (n8n9);
NetDeviceContainer d9d0 = p2p.Install (n9n0);
NetDeviceContainer d1d5 = p2p.Install (n1n5);
NetDeviceContainer d6d9 = p2p.Install (n6n9);

// С помощью Ipv4AddressHelper раздаем адреса IPv4 устройствам
// С помощью Ipv4InterfaceContainer будем обращаться к адресам устройств
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer i0i1 = ipv4.Assign (d0d1);

ipv4.SetBase ("10.1.2.0", "255.255.255.0");
ipv4.Assign (d1d2);

ipv4.SetBase ("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer i2i3 = ipv4.Assign (d2d3);

ipv4.SetBase ("10.1.4.0", "255.255.255.0");
ipv4.Assign (d3d4);

ipv4.SetBase ("10.1.5.0", "255.255.255.0");
ipv4.Assign (d4d5);

ipv4.SetBase ("10.1.6.0", "255.255.255.0");
Ipv4InterfaceContainer i5i6 = ipv4.Assign (d5d6);
```

```
	ipv4.SetBase ("10.1.7.0", "255.255.255.0");
	ipv4.Assign (d6d7);

	ipv4.SetBase ("10.1.8.0", "255.255.255.0");
	ipv4.Assign (d7d8);

	ipv4.SetBase ("10.1.9.0", "255.255.255.0");
	Ipv4InterfaceContainer i8i9 = ipv4.Assign (d8d9);

	ipv4.SetBase ("10.1.0.0", "255.255.255.0");
	ipv4.Assign (d9d0);

	ipv4.SetBase ("10.1.11.0", "255.255.255.0");
	ipv4.Assign (d1d5);

	ipv4.SetBase ("10.1.12.0", "255.255.255.0");
	ipv4.Assign (d6d9);

	// Создание узлов маршрутизации, инициализация базы данных
маршрутизации,
	// настройка таблицы маршрутизации в узлах
	Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

	// Создание и настройка приложения, которое будет принимать пакеты
	uint16_t port = 9;
	PacketSinkHelper sink ("ns3::UdpSocketFactory",
	                       Address (InetSocketAddress (Ipv4Address::GetAny (),
port)));
	ApplicationContainer sinkApp = sink.Install (c.Get (6));
	sinkApp.Start (Seconds (0.0));
	sinkApp.Stop (Seconds (6.0));

	PacketSinkHelper sink2 ("ns3::UdpSocketFactory",
	                        Address (InetSocketAddress (Ipv4Address::GetAny (),
port)));
	ApplicationContainer sinkApp2 = sink2.Install (c.Get (8));
	sinkApp2.Start (Seconds (0.0));
	sinkApp2.Stop (Seconds (6.0));

	PacketSinkHelper sink3 ("ns3::UdpSocketFactory",
	                        Address (InetSocketAddress (Ipv4Address::GetAny (),
port)));
	ApplicationContainer sinkApp3 = sink3.Install (c.Get (9));
```

```
sinkApp3.Start (Seconds (0.0));
sinkApp3.Stop (Seconds (6.0));

// Создание и настройка приложения, которое будет генерировать и
отправлять пакеты на сервер
OnOffHelper clientHelper ("ns3::UdpSocketFactory",
                InetSocketAddress (i5i6.GetAddress (1), port));
clientHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
clientHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
clientHelper.SetAttribute ("DataRate", StringValue ("2kbps"));
clientHelper.SetAttribute ("PacketSize", UintegerValue (50));

// Создаем клиента при помощи clientHelper, который используется для
связи с сервером
ApplicationContainer apps = clientHelper.Install (c.Get (1));
apps.Start (Seconds (0.5));
apps.Stop (Seconds (6.0));

OnOffHelper clientHelper2 ("ns3::UdpSocketFactory",
                InetSocketAddress (i8i9.GetAddress (0), port));
clientHelper2.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
clientHelper2.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
clientHelper2.SetAttribute ("DataRate", StringValue ("2kbps"));
clientHelper2.SetAttribute ("PacketSize", UintegerValue (50));

ApplicationContainer apps2 = clientHelper2.Install (c.Get (0));
apps2.Start (Seconds (0.5));
apps2.Stop (Seconds (6.0));

OnOffHelper clientHelper3 ("ns3::UdpSocketFactory",
                InetSocketAddress (i8i9.GetAddress (1), port));
clientHelper3.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
clientHelper3.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
clientHelper3.SetAttribute ("DataRate", StringValue ("2kbps"));
clientHelper3.SetAttribute ("PacketSize", UintegerValue (50));

ApplicationContainer apps3 = clientHelper3.Install (c.Get (3));
```

```
apps3.Start (Seconds (0.5));
apps3.Stop (Seconds (6.0));

// Ниже происходит обрыв связей через 1.5 секунды после начала моделирования
// Связь восстанавливается через 4 секунды после начала моделирования
// У каждого узла может быть несколько интерфейсов
// Первый - loopback пронумерован 0, затем идет интерфейс 1 для перовой связи p2p,
// затем интерфейс 2 для следующей p2p связи и т.д.
// Ниже убираем связь между узлом 1 и узлом 5, а это 3-я p2p связь для узла 1
Ptr<Node> n1 = c.Get (1);
Ptr<Ipv4> ipv41 = n1->GetObject<Ipv4> ();
Simulator::Schedule (Seconds (1.5), &Ipv4::SetDown, ipv41, 3);
Simulator::Schedule (Seconds (4.0), &Ipv4::SetUp, ipv41, 3);

Ptr<Node> n2 = c.Get (2);
Ptr<Ipv4> ipv42 = n2->GetObject<Ipv4> ();
Simulator::Schedule (Seconds (1.5), &Ipv4::SetDown, ipv42, 2);
Simulator::Schedule (Seconds (4.0), &Ipv4::SetUp, ipv42, 2);

Ptr<Node> n8 = c.Get (8);
Ptr<Ipv4> ipv48 = n8->GetObject<Ipv4> ();
Simulator::Schedule (Seconds (1.5), &Ipv4::SetDown, ipv48, 2);
Simulator::Schedule (Seconds (4.0), &Ipv4::SetUp, ipv48, 2);

// Трейс-файл общего вида. Файл открывается с помощью TraceMetrics
AsciiTraceHelper ascii;
p2p.EnableAsciiAll (ascii.CreateFileStream ("lab2.tr"));

// Файл для визуализатора NetAnim, а также установка положения узлов
AnimationInterface anim("animLab2.xml");
anim.SetConstantPosition(c.Get(0), 5.0, 0.0);
anim.SetConstantPosition(c.Get(1), 6.0, 1.0);
anim.SetConstantPosition(c.Get(2), 7.0, 2.0);
anim.SetConstantPosition(c.Get(3), 6.0, 3.0);
anim.SetConstantPosition(c.Get(4), 5.0, 4.0);
anim.SetConstantPosition(c.Get(5), 4.0, 5.0);
anim.SetConstantPosition(c.Get(6), 3.0, 4.0);
anim.SetConstantPosition(c.Get(7), 2.0, 3.0);
```

```
  anim.SetConstantPosition(c.Get(8), 3.0, 2.0);
  anim.SetConstantPosition(c.Get(9), 4.0, 1.0);

  Simulator::Stop (Seconds(6.0));
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

10.3. Пример 3

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"
#include "ns3/point-to-point-star.h"
#include "ns3/csma-module.h"

using namespace ns3;

int main (int argc, char *argv[])
{
  int nCsma = 2;
  Time::SetResolution (Time::NS);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
  pointToPoint.SetQueue ("ns3::RedQueue");

  PointToPointStarHelper p2pStar(5, pointToPoint);

  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", StringValue ("10Mbps"));
  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
  csma.SetQueue ("ns3::RedQueue");

  NodeContainer csmaN1;
  csmaN1.Add (p2pStar.GetSpokeNode(0));
  csmaN1.Create (nCsma);
```

```
NodeContainer csmaN2;
csmaN2.Add (p2pStar.GetSpokeNode(2));
csmaN2.Create (nCsma);

NetDeviceContainer csmaD1;
csmaD1 = csma.Install (csmaN1);

NetDeviceContainer csmaD2;
csmaD2 = csma.Install (csmaN2);

InternetStackHelper stack;
p2pStar.InstallStack(stack);
stack.Install(csmaN1.Get(1));
stack.Install(csmaN1.Get(2));
stack.Install(csmaN2.Get(1));
stack.Install(csmaN2.Get(2));

Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
p2pStar.AssignIpv4Addresses(address);

address.SetBase ("172.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer csmaI1;
csmaI1 = address.Assign (csmaD1);

address.SetBase ("172.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaI2;
csmaI2 = address.Assign (csmaD2);


//Turn on global static routing
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();


// Create a packet sink on the star "hub" to receive these packets
uint16_t port = 50000;
Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (),
port));
PacketSinkHelper sinkHelper ("ns3::UdpSocketFactory",
sinkLocalAddress);
ApplicationContainer sinkApp = sinkHelper.Install (csmaN1.Get(2));
sinkApp.Start (Seconds (1.0));
sinkApp.Stop (Seconds (10.0));
```

```
    // Create the OnOff applications to send TCP to the server
    OnOffHelper clientHelper ("ns3::UdpSocketFactory", Address ());
    clientHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
    clientHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));

    ApplicationContainer clientApps;
    AddressValue remoteAddress (InetSocketAddress (csmaI1.GetAddress(2),
port));
    clientHelper.SetAttribute ("Remote", remoteAddress);
    clientHelper.SetAttribute ("PacketSize", UintegerValue (210));
    clientApps.Add (clientHelper.Install (csmaN2.Get(2)));
    clientApps.Start (Seconds (1.0));
    clientApps.Stop (Seconds (10.0));


    ApplicationContainer sinkApp2 = sinkHelper.Install (csmaN2.Get(1));
    sinkApp2.Start (Seconds (1.0));
    sinkApp2.Stop (Seconds (10.0));

    ApplicationContainer clientApps2;
    AddressValue remoteAddress2 (InetSocketAddress (csmaI2.GetAddress(1),
port));
    clientHelper.SetAttribute ("Remote", remoteAddress2);
    clientApps2.Add (clientHelper.Install (csmaN1.Get(1)));
    clientApps2.Start (Seconds (1.0));
    clientApps2.Stop (Seconds (10.0));


    AnimationInterface anim("animLab3.xml");

    anim.SetConstantPosition(p2pStar.GetHub(), 6.0, 10.0);
    anim.SetConstantPosition(p2pStar.GetSpokeNode(0), 2.0, 5.0);
    anim.SetConstantPosition(p2pStar.GetSpokeNode(1), 10.0, 5.0);
    anim.SetConstantPosition(p2pStar.GetSpokeNode(2), 6.0, 15.0);
    anim.SetConstantPosition(p2pStar.GetSpokeNode(3), 0.0, 15.0);
    anim.SetConstantPosition(p2pStar.GetSpokeNode(4), 12.0, 15.0);

    anim.SetConstantPosition(csmaN1.Get(1), 0.0, 3.0);
    anim.SetConstantPosition(csmaN1.Get(2), 4.0, 3.0);
```

```
  anim.SetConstantPosition(csmaN2.Get(1), 4.0, 17.0);
  anim.SetConstantPosition(csmaN2.Get(2), 8.0, 17.0);

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

10.4. Пример 4

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"
#include "ns3/point-to-point-star.h"
#include "ns3/stats-module.h"

using namespace ns3;

int main (int argc, char *argv[])
{
 Time::SetResolution (Time::NS);

  // Создаем канал, который используется для связи между узлами
  // Для этого используем класс PointToPointHelper, с помощью которого
указываем
  // параметры канала
  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("2Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
  pointToPoint.SetQueue ("ns3::DropTailQueue",
"MaxPackets",UintegerValue (2));

  // С помощью PointToPointStarHelper создается топология "звезда"
  // В данном случае имеем 5 узлов, которые связаны с центральным
узлом
  // посредством каналов pointToPoint
  PointToPointStarHelper p2pStar(5, pointToPoint);

  // Ниже создается контейнере в котором цетральный узел "звезды"
плюс дополнительный узел
```

```
NodeContainer c;
c.Add(p2pStar.GetHub());
c.Create(1);

// Устанавливаем сетевые стеки на узлах
InternetStackHelper stack;
p2pStar.InstallStack(stack);
stack.Install(c.Get(1));

// Устанавливаем устройства и канал для центрального узла "звезды" и дополнительного узла
NetDeviceContainer d = pointToPoint.Install (c);

// Далее раздаем IP-адреса всем устройствам
// С помощью Ipv4InterfaceContainer мы будем обращаться к адресу устройств
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer i = address.Assign (d);

address.SetBase ("10.1.2.0", "255.255.255.0");
p2pStar.AssignIpv4Addresses(address);

// Создание узлов маршрутизации, инициализация базы данных маршрутизации,
// настройка таблицы маршрутизации в узлах
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

// Создание и настройка приложения, которое будет принимать пакеты
uint16_t port = 50000;
Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper sinkHelper ("ns3::UdpSocketFactory", sinkLocalAddress);
ApplicationContainer sinkApp = sinkHelper.Install (c.Get(1));
sinkApp.Start (Seconds (0.0));
sinkApp.Stop (Seconds (10.0));

// Создаем приложение которое будет генерировать и отправлять пакеты на сервер
OnOffHelper clientHelper ("ns3::UdpSocketFactory", Address ());
clientHelper.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
```

```
    clientHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
    clientHelper.SetAttribute ("DataRate", StringValue ("1Mbps"));
    clientHelper.SetAttribute ("PacketSize", UintegerValue (64000));

    // Создаем клиент на каждом внешнем узле "звезды" при помощи
clientHelper,
    // который используется для связи с сервером
    ApplicationContainer clientApps;
    AddressValue remoteAddress (InetSocketAddress (i.GetAddress(1), port));
    clientHelper.SetAttribute ("Remote", remoteAddress);
    clientApps.Add (clientHelper.Install (p2pStar.GetSpokeNode(0)));
    clientApps.Add (clientHelper.Install (p2pStar.GetSpokeNode(1)));
    clientApps.Add (clientHelper.Install (p2pStar.GetSpokeNode(2)));
    clientApps.Add (clientHelper.Install (p2pStar.GetSpokeNode(3)));
    clientApps.Add (clientHelper.Install (p2pStar.GetSpokeNode(4)));
    clientApps.Start (Seconds (1.0));
    clientApps.Stop (Seconds (10.0));

    // Создаем файл для анимации. Затем устанавливаем позиции(X и Y)
для каждого узла.
    AnimationInterface anim("animLab4.xml");
    anim.SetConstantPosition(c.Get(1), 15.0, 6.0);
    anim.SetConstantPosition(p2pStar.GetHub(), 10.0, 6.0);
    anim.SetConstantPosition(p2pStar.GetSpokeNode(0), 5.0, 2.0);
    anim.SetConstantPosition(p2pStar.GetSpokeNode(1), 5.0, 4.0);
    anim.SetConstantPosition(p2pStar.GetSpokeNode(2), 5.0, 6.0);
    anim.SetConstantPosition(p2pStar.GetSpokeNode(3), 5.0, 8.0);
    anim.SetConstantPosition(p2pStar.GetSpokeNode(4), 5.0, 10.0);

    // Трейс-файл общего вида
    AsciiTraceHelper ascii;
    pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("lab4.tr"));

    // Следующая часть кода устанавливает время остановки симуляции,
    // затем запускает симулятор, убирает его и выполняет выход из
программы
    Simulator::Stop (Seconds (10.0));
    Simulator::Run ();
    Simulator::Destroy ();
    return 0;
  }
```

10.5. Пример 5

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

using namespace ns3;
int
main (int argc, char *argv[])
{
  // Создание контейнера для 8 узлов
  NodeContainer n;
  n.Create (8);

  // Ниже будут создаваться wifi-устройства и канал связи между узлами
  // Сначала необходимо настроить помощники для физического уровня и канального уровня
  // В данном случае используются настройки по-умолчанию
  YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
  YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();

  // После того, как оба объекта созданы,
  // мы ассоциируем объект канального уровня с объектом физического уровня
  phy.SetChannel (channel.Create ());

  // Метод SetRemoteStationManager определяет алгоритм регулирования скорости
  // В данном случае - это AARF алгоритм
  WifiHelper wifi = WifiHelper::Default ();
  wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

  // NqosWifiMacHelper устанавливает параметры MAC-адресов. В данном случае по-умолчанию
  NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();

  // Далее указываем типа MAC-адресов, идентификатор сети
  Ssid ssid = Ssid ("ns-3-ssid");
  mac.SetType ("ns3::AdhocWifiMac",
```

```
                "Ssid", SsidValue (ssid));

        // Устанавливаем устройства
        NetDeviceContainer d;
        d = wifi.Install (phy, mac, n);

        // Далее необходимо определить какие узлы будут двигаться, а какие
будут иметь постоянную позицию
        // Для этого используем MobilityHelper
        MobilityHelper mobility;
        // Этот код определяет исползование двумерной сетки для размещения
узлов
        mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                       "MinX", DoubleValue (0.0),
                                       "MinY", DoubleValue (0.0),
                                       "DeltaX", DoubleValue (5.0),
                                       "DeltaY", DoubleValue (10.0),
                                       "GridWidth", UintegerValue (3),
                                       "LayoutType", StringValue ("RowFirst"));
        // Указываем те узлы, которые будут двагаться
        mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
                                   "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));
        mobility.Install (n.Get(0));
        mobility.Install (n.Get(3));
        mobility.Install (n.Get(5));
        mobility.Install (n.Get(6));

        // Указываем те узлы, которые будут иметь постоянную позицию
        mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
        mobility.Install (n.Get(1));
        mobility.Install (n.Get(2));
        mobility.Install (n.Get(4));
        mobility.Install (n.Get(7));

        // Указываем протокол маршрутизыации и устанавливаем сетевые
стеки на узлы
        AodvHelper aodv;
        InternetStackHelper stack;
        stack.SetRoutingHelper (aodv);
        stack.Install (n);

        // Раздаем узлам адреса IPv4
        Ipv4AddressHelper address;
```

```
address.SetBase ("10.1.1.0", "255.255.255.0");

// Определяем объект, с помощью которого будем обращаться к Ip-адресам
Ipv4InterfaceContainer ipv4;
ipv4 = address.Assign (d);

// Создание и настройка приложения, которое будет принимать пакеты
uint16_t port = 49877;
Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper sinkHelper ("ns3::UdpSocketFactory", sinkLocalAddress);
ApplicationContainer sinkApp = sinkHelper.Install (n.Get (7));
sinkApp.Start (Seconds (0.0));
sinkApp.Stop (Seconds (10.0));

// Создаем приложение которое будет генерировать и отправлять пакеты на сервер
OnOffHelper clientHelper ("ns3::UdpSocketFactory", Address ());
clientHelper.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
clientHelper.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
clientHelper.SetAttribute ("DataRate", StringValue ("500kbps"));
clientHelper.SetAttribute ("PacketSize", UintegerValue (200));

// Создаем клиента при помощи clientHelper, который используется для связи с сервером
ApplicationContainer clientApps;
AddressValue remoteAddress (InetSocketAddress (ipv4.GetAddress (7), port));
clientHelper.SetAttribute ("Remote", remoteAddress);
clientApps.Add (clientHelper.Install (n.Get (0)));
clientApps.Start (Seconds (0.5));
clientApps.Stop (Seconds (10.0));


ApplicationContainer sinkApp2 = sinkHelper.Install (n.Get (5));
sinkApp2.Start (Seconds (0.0));
sinkApp2.Stop (Seconds (10.0));
```

```
    ApplicationContainer clientApps2;
    AddressValue remoteAddress2 (InetSocketAddress (ipv4.GetAddress (5),
port));
    clientHelper.SetAttribute ("Remote", remoteAddress2);
    clientApps2.Add (clientHelper.Install (n.Get (2)));
    clientApps2.Start (Seconds (0.5));
    clientApps2.Stop (Seconds (10.0));

    // Создание узлов маршрутизации, инициализация базы данных
маршрутизации,
    // настройка таблицы маршрутизации в узлах
    Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

    // Создаем файл для анимации. Позиции для узлов были указаны с
помощью MobilityHelper
    AnimationInterface anim("animLab5.xml");

    // Трейс-файл общего вида
    AsciiTraceHelper ascii;
    phy.EnableAsciiAll (ascii.CreateFileStream ("lab5.tr"));

    // Следующая часть кода устанавливает время остановки симуляции,
    // затем запускает симулятор, убирает его и выполняет выход из
программы
    Simulator::Stop (Seconds (10.0));
    Simulator::Run ();
    Simulator::Destroy ();
    return 0;
}
```

10.6. Пример 6

```
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv6-static-routing-helper.h"
#include "ns3/ipv6-routing-table-entry.h"
#include "ns3/sixlowpan-module.h"
#include "ns3/netanim-module.h"
#include "ns3/mobility-module.h"
#include "ns3/lr-wpan-module.h"

using namespace ns3;
```

```
int main (int argc, char** argv)
{

  // Создание двух узлов
  Ptr<Node> n0 = CreateObject<Node> ();
  Ptr<Node> n1 = CreateObject<Node> ();

  // Контейнер для всех узлов
  NodeContainer c (n0, n1);

  // С помощью MobilityHelper указываем позицию для узлов
  MobilityHelper mobility;
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                 "MinX", DoubleValue (0.0), "MinY", DoubleValue
(0.0),
                                 "DeltaX", DoubleValue (10.0), "DeltaY", DoubleValue
(10.0),
                                 "GridWidth", UintegerValue (3), "LayoutType",
StringValue ("RowFirst"));
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (c);

  // Создаем low-rate wireless personal area network, устанавливаем
устройства для каждого узла
  LrWpanHelper lrWpanHelper;
  NetDeviceContainer d = lrWpanHelper.Install(c);

  // Ассоциация устройств для Personal Area Network и короткое
назначение адресов
  lrWpanHelper.AssociateToPan (d, 0);

  // Устанавливаем сетевые стеки для узлов
  InternetStackHelper stack;
  stack.Install(c);

  // Устанавливаем стек 6LowPan на устройства
  SixLowPanHelper sixlowpan;
  NetDeviceContainer slpD = sixlowpan.Install (d);

  // Далее раздаем IP-адреса всем устройствам
  // С помощью Ipv6InterfaceContainer мы будем обращаться к адресам
устройств
```

```
Ipv6AddressHelper ipv6;
ipv6.SetBase (Ipv6Address ("2001:1::"), Ipv6Prefix (64));
Ipv6InterfaceContainer i = ipv6.Assign (slpD);

// Создание и настройка приложения, которое будет принимать пакеты
uint16_t port = 50000;
Address sinkLocalAddress (Inet6SocketAddress (i.GetAddress(0, 1), port));
PacketSinkHelper sinkHelper ("ns3::UdpSocketFactory",
sinkLocalAddress);
ApplicationContainer sinkApp = sinkHelper.Install (n1);
sinkApp.Start (Seconds (1.0));
sinkApp.Stop (Seconds (10.0));

// Создаем приложение которое будет генерировать и отправлять
пакеты на сервер
OnOffHelper clientHelper ("ns3::UdpSocketFactory", Address ());
clientHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
clientHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));

// Создаем клиента при помощи clientHelper, который используется для
связи с сервером
ApplicationContainer clientApps;
AddressValue remoteAddress (Inet6SocketAddress (i.GetAddress(1, 1),
port));
clientHelper.SetAttribute ("Remote", remoteAddress);
clientApps.Add (clientHelper.Install (n0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));

/*
 *Еще один вариант отправки пакетов между устройствами

uint32_t packetSize = 10;
uint32_t maxPacketCount = 50;
Ping6Helper ping6;

ping6.SetLocal (i.GetAddress (0, 1));
ping6.SetRemote (i.GetAddress (1, 1));

ping6.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
ping6.SetAttribute ("PacketSize", UintegerValue (packetSize));
```

```
    ApplicationContainer apps = ping6.Install (n0);

    apps.Start (Seconds (0.5));
    apps.Stop (Seconds (15.0));*/

    // Создаем файл для анимации
    AnimationInterface anim("animLab6.xml");

    // Трейс-файл общего вида
    AsciiTraceHelper ascii;
    lrWpanHelper.EnableAsciiAll (ascii.CreateFileStream ("lab6.tr"));

    // Следующая часть кода устанавливает время остановки симуляции,
    // затем запускает симулятор, убирает его и выполняет выход из
программы
    Simulator::Stop (Seconds (15));
    Simulator::Run ();
    Simulator::Destroy ();

  }
```