Сетевой стек Reticulum

*Выпуск **1.0.1***

Марк Квист

02 ноября 2025 г.

СОДЕРЖАНИЕ

1 Что такое Reticulum? **3**
- 1.1 Текущий статус . . . 3
- 1.2 Что предлагает Reticulum? . . . 3
- 1.3 Где может применяться Reticulum? . . . 4
- 1.4 Типы интерфейсов и устройств . . . 5
- 1.5 Caveat Emptor . . . 5

2 Быстрый старт **7**
- 2.1 Автономная установка Reticulum . . . 7
  - 2.1.1 Устранение проблем с зависимостями и установкой . . . 7
- 2.2 Попробуйте использовать программу на основе Reticulum . . . 8
  - 2.2.1 Удаленная оболочка . . . 8
  - 2.2.2 Nomad Network . . . 8
  - 2.2.3 Sideband . . . 9
  - 2.2.4 MeshChat . . . 10
- 2.3 Использование входящих в комплект утилит . . . 10
- 2.4 Создание сети с помощью Reticulum . . . 10
- 2.5 Соединение экземпляров Reticulum через Интернет . . . 11
- 2.6 Подключение к публичной тестовой сети . . . 11
- 2.7 Размещение публичных точек входа . . . 12
- 2.8 Добавление радиоинтерфейсов . . . 13
- 2.9 Создание и использование пользовательских интерфейсов . . . 13
- 2.10 Разработка программы с использованием Reticulum . . . 13
- 2.11 Участие в разработке Reticulum . . . 14
- 2.12 Примечания по установке для конкретных платформ . . . 14
  - 2.12.1 Android . . . 14
  - 2.12.2 ARM64 . . . 15
  - 2.12.3 Debian Bookworm . . . 16
  - 2.12.4 MacOS . . . 16
  - 2.12.5 OpenWRT . . . 17
  - 2.12.6 Raspberry Pi . . . 18
  - 2.12.7 RISC-V . . . 18
  - 2.12.8 Ubuntu Lunar . . . 18
  - 2.12.9 Windows . . . 19
- 2.13 Pure-Python Reticulum . . . 19

3 Использование Reticulum в вашей системе **21**
- 3.1 Конфигурация и данные . . . 21
- 3.2 Включенные служебные программы . . . 24
  - 3.2.1 Утилита rnsd . . . 25

3.2.2 Утилита rnstatus . . . 25
3.2.3 Утилита rnid . . . 27
3.2.4 Утилита rnpath . . . 29
3.2.5 Утилита rnprobe . . . 30
3.2.6 утилита rncp . . . 31
3.2.7 Утилита rnx . . . 32
3.2.8 Утилита rnodeconf . . . 33
3.3 Удалённое управление . . . 34
3.4 Улучшение конфигурации системы . . . 35
3.4.1 Исправленные имена последовательных портов . . . 35
3.4.2 Reticulum как системная служба . . . 35

4. Понимание Reticulum **39**
4.1 Мотивация . . . 39
4.2 Цели . . . 40
4.3 Введение и базовая функциональность . . . 40
4.3.1 Назначения . . . 41
4.3.2 Объявления открытых ключей . . . 43
4.3.3 Идентификаторы . . . 43
4.3.4 Дальнейшие шаги . . . 44
4.4 Транспорт Reticulum . . . 44
4.4.1 Типы узлов . . . 44
4.4.2 Механизм объявлений подробно . . . 44
4.4.3 Достижение пункта назначения . . . 45
4.4.4 Ресурсы . . . 47
4.5 Эталонная установка . . . 47
4.6 Специфика протокола . . . 48
4.6.1 Приоритизация пакетов . . . 49
4.6.2 Коды доступа к интерфейсу . . . 49
4.6.3 Формат канала . . . 49
4.6.4 Правила распространения объявлений . . . 51
4.6.5 Криптографические примитивы . . . 52

5 Аппаратные средства связи **55**
5.1 Комбинирование типов оборудования . . . 55
5.2 RNode . . . 55
5.2.1 Создание RNodes . . . 56
5.2.2 Поддерживаемые платы и устройства . . . 56
5.2.3 Установка . . . 62
5.2.4 Использование с Reticulum . . . 62
5.3 Оборудование на базе Wi-Fi . . . 62
5.4 Оборудование на базе Ethernet . . . 63
5.5 Последовательные линии и устройства . . . 63
5.6 Пакетные радиомодемы . . . 63

6 Настройка интерфейсов **65**
6.1 Пользовательские интерфейсы . . . 65
6.2 Автоматический интерфейс . . . 65
6.3 Магистральный интерфейс . . . 67
6.3.1 Слушатели . . . 67
6.3.2 Подключение удаленных устройств . . . 68
6.4 Интерфейс TCP-сервера . . . 68
6.5 Интерфейс TCP-клиента . . . 70
6.6 Интерфейс UDP . . . 71

6.7 Интерфейс I2P . . . 72
6.8 Интерфейс RNode LoRa . . . 73
6.9 Интерфейс RNode Multi . . . 74
6.10 Последовательный интерфейс . . . 77
6.11 Интерфейс Pipe . . . 77
6.12 Интерфейс KISS . . . 77
6.13 Интерфейс AX.25 KISS . . . 79
6.14 Общие параметры интерфейса . . . 80
6.15 Режимы интерфейса . . . 81
6.16 Контроль скорости объявлений . . . 82
6.17 Ограничение скорости для новых назначений . . . 82

7 Построение сетей **85**
7.1 Концепции и обзор . . . 85
7.2 Примеры сценариев . . . 86
7.2.1 Взаимосвязанные объекты LoRa . . . 86
7.2.2 Мост через Интернет . . . 87
7.2.3 Рост и конвергенция . . . 87

8 Поддержка Reticulum **89**
8.1 Пожертвования . . . 89
8.2 Обеспечить обратную связь . . . 89
8.3 Внести код . . . 89

9 Примеры кода **91**
9.1 Минимальный . . . 91
9.2 Объявление . . . 93
9.3 Широковещание . . . 97
9.4 Эхо . . . 99
9.5 Связь . . . 106
9.6 Идентификация . . . 111
9.7 Запросы и ответы . . . 118
9.8 Канал . . . 123
9.9 Буфер . . . 131
9.10 Передача файлов . . . 137
9.11 Пользовательские интерфейсы . . . 149

10 Справочник API **157**
10.1 Reticulum . . . 157
10.2 Идентичность . . . 158
10.3 Назначение . . . 162
10.4 Пакет . . . 166
10.5 Получение пакетов . . . 167
10.6 Связь . . . 167
10.7 Получение запросов . . . 171
10.8 Ресурс . . . 172
10.9 Канал . . . 173
10.10 MessageBase . . . 174
10.11 Буфер . . . 174
10.12 RawChannelReader . . . 175
10.13 RawChannelWriter . . . 176
10.14 Транспорт . . . 176

Индекс **179**

Цель данного руководства — предоставить вам всю необходимую информацию для понимания Reticulum, создания сетей или разработки программ с его использованием, а также для участия в разработке самого Reticulum.

ГЛАВА

ПЕРВАЯ

ЧТО ТАКОЕ RETICULUM?

Reticulum — это основанный на криптографии сетевой стек для создания как локальных, так и глобальных сетей с использованием общедоступного оборудования, который может продолжать функционировать в неблагоприятных условиях, таких как крайне низкая пропускная способность и очень высокая задержка.

Reticulum позволяет создавать глобальные сети с помощью готовых инструментов и предлагает сквозное шифрование, прямую секретность, автоконфигурируемый криптографически защищенный многоузловой транспорт, эффективную адресацию, неподдельные подтверждения пакетов и многое другое.

С точки зрения пользователя, Reticulum позволяет создавать приложения, которые уважают и расширяют автономию и суверенитет сообществ и отдельных лиц. Reticulum обеспечивает безопасную цифровую связь, которая не может быть подвергнута внешнему контролю, манипуляциям или цензуре.

Reticulum позволяет строить как небольшие, так и потенциально планетарные сети без какой-либо необходимости в иерархических или бюрократических структурах для их контроля или управления, обеспечивая при этом полную суверенность отдельных лиц и сообществ над их собственными сетевыми сегментами.

Reticulum — это **полный сетевой стек** , которому не требуется IP или более высокие уровни, хотя его легко использовать (с TCP или UDP) в качестве базового носителя для Reticulum. Поэтому тривиально туннелировать Reticulum через Интернет или частные IP-сети. Reticulum построен непосредственно на криптографических принципах, что обеспечивает отказоустойчивость и стабильную функциональность в открытых и недоверенных сетях

Не требуется никаких модулей ядра или драйверов. Reticulum может работать полностью в пользовательском пространстве и будет функционировать практически на любой системе, где запущен Python 3. Reticulum хорошо работает даже на небольших одноплатных компьютерах, таких как Pi Zero.

1.1 Текущий статус

Все основные функции протокола реализованы и функционируют, но, вероятно, будут дополнения по мере изучения реального использования . API и формат передачи данных можно считать полными и стабильными, но они могут измениться, если это будет абсолютно оправдано.

1.2 Что предлагает Reticulum?

- Глобально уникальная адресация и идентификация без координации.
- Полностью самонастраивающаяся многохоповая маршрутизация по гетерогенным носителям.
- Гибкая масштабируемость по гетерогенным топологиям
  - Reticulum может передавать данные по любой комбинации физических сред и топологий.
  - Низкоскоростные сети могут сосуществовать и взаимодействовать с большими, высокоскоростными сетями.
- Анонимность инициатора, общение без раскрытия вашей личности
  - Reticulum не включает исходные адреса в какие-либо пакеты.
- Асимметричное шифрование X25519 и подписи Ed25519 как основа для всей связи.

  - Основополагающие ключи идентификации Reticulum представляют собой 512-битные наборы ключей эллиптической кривой.
- Прямая секретность доступна для всех типов связи, как для отдельных пакетов, так и по каналам.
- Reticulum использует следующий формат для зашифрованных токенов:
  - Эфемерные ключи для каждого пакета и канала, полученные из обмена ключами ECDH на Curve25519
  - AES-256 в режиме CBC с дополнением PKCS7
  - HMAC с использованием SHA256 для аутентификации
  - IVs генерируются с помощью os.urandom()
- Неподделываемые подтверждения доставки пакетов
- Гибкая и расширяемая система интерфейсов
  - Reticulum включает большое разнообразие встроенных типов интерфейсов
  - Возможность загружать и использовать пользовательские или предоставленные сообществом типы интерфейсов
  - Легко создавать свои собственные интерфейсы для связи по чему угодно
- Аутентификация и сегментация виртуальных сетей на всех поддерживаемых типах интерфейсов
- Интуитивно понятный и простой в использовании API
  - Проще и легче использовать, чем API сокетов; проще, но мощнее
  - Значительно упрощает создание распределенных и децентрализованных приложений
- Надежная и эффективная передача произвольных объемов данных
  - Reticulum может обрабатывать несколько байтов данных или файлы размером в несколько гигабайт
  - Упорядочивание, сжатие, координация передачи и контрольные суммы выполняются автоматически
  - API очень прост в использовании и предоставляет информацию о ходе передачи
- Легковесный, гибкий и расширяемый механизм Запрос/Ответ
- Эффективное установление соединения
  - Общая стоимость установки зашифрованного и верифицированного соединения составляет всего 3 пакета, общим объемом 297 байт
  - Низкая стоимость поддержания открытых соединений — всего 0,44 бита в секунду
- Надежная последовательная доставка с помощью механизмов Channel и Buffer

1.3 Где можно использовать Reticulum?

Практически в любой среде, которая может поддерживать как минимум полудуплексный канал с пропускной способностью более 5 бит в секунду и MTU 500 байт. Радиостанции для передачи данных, модемы, LoRa-радио, последовательные линии, AX.25 TNC, цифровые режимы любительской радиосвязи, ad-hoc WiFi, оптические линии свободного пространства и аналогичные системы — все это примеры типов интерфейсов, для которых был разработан Reticulum.

Интерфейс с открытым исходным кодом на базе LoRa под названием RNode был разработан как пример приемопередатчика, очень подходящего для Reticulum. Его можно собрать самостоятельно, преобразовать обычную плату разработки LoRa в такой приемопередатчик, или его можно приобрести как полноценный приемопередатчик у различных поставщиков.

Reticulum также может быть инкапсулирован поверх существующих IP-сетей, поэтому ничто не мешает использовать его через проводной Ethernet или вашу локальную сеть WiFi, где он будет работать так же хорошо. На самом деле, одна из сильных сторон Reticulum заключается в том, насколько легко он позволяет подключать различные среды в самоконфигурируемую, отказоустойчивую и зашифрованную mesh-сеть.

Например, можно настроить Raspberry Pi, подключенный как к радио LoRa, так и к TNC пакетного радио и Сети WiFi. После добавления интерфейсов Reticulum позаботится об остальном, и любое устройство в Сети WiFi сможет связываться с узлами на сторонах LoRa и пакетного радио сети, и наоборот.

1.4 Типы интерфейсов и устройств

Reticulum реализует ряд обобщенных типов интерфейсов, которые охватывают коммуникационное оборудование, поверх которого может работать Reticulum. Если ваше оборудование не поддерживается, легко *реализовать класс интерфейса* . В настоящее время Reticulum может использовать следующие устройства и средства связи:

- Любое устройство Ethernet
  - Устройства Wi-Fi
  - Проводные устройства Ethernet
  - Волоконно-оптические приемопередатчики
  - Радиостанции данных с портами Ethernet
- LoRa с использованием RNode
  - Может быть установлено на многих популярных платах LoRa
  - Может быть приобретено как готовый к использованию приемопередатчик
- Пакетные радио-TNC, такие как OpenModem
  - Любой пакетный радио-TNC в режиме KISS
  - Идеально подходит для VHF- и UHF-радио
- Любое устройство с последовательным портом
- Сеть I2P
- TCP поверх IP-сетей
- UDP поверх IP-сетей
- Всё, что можно подключить через stdio
  - Reticulum может использовать внешние программы и каналы в качестве интерфейсов
  - Это может быть использовано для легкого взлома виртуальных интерфейсов.
  - Или для быстрого создания интерфейсов с пользовательским оборудованием.

Полный список и более подробную информацию см. в главе*«Поддерживаемые интерфейсы»* .

1.5 Осторожно, покупатель

Reticulum — это экспериментальный сетевой стек, и его следует рассматривать именно так. Хотя он был разработан с учетом передовых практик криптографии, он еще не прошел внешний аудит безопасности, и вполне возможно наличие ошибок, нарушающих конфиденциальность. Чтобы считаться безопасным, Reticulum нуждается в тщательном аудите безопасности независимыми криптографами и исследователями безопасности. Если вы хотите помочь в этом или можете спонсировать аудит, пожалуйста, свяжитесь с нами

ГЛАВА

ДВА

БЫСТРЫЙ СТАРТ

Лучший способ начать работу с сетевым стеком Reticulum зависит от того, что вы хотите делать . Это руководство описывает разумные начальные пути для различных сценариев.

2.1 Автономная установка Reticulum

Если вы просто хотите установить Reticulum и связанные с ним утилиты в систему, самый простой способ — через менеджер пакетов `pip`:

```
pip install rns
```

Если у вас еще не установлен pip, вы можете установить его с помощью системного менеджера пакетов командой вроде `sudo apt install python3-pip` , `sudo pamac install python-pip` или аналогичной.

Вы также можете скачать релизные файлы Reticulum (колеса) с GitHub или других каналов выпуска и установить их в автономном режиме с помощью `pip` :

```
pip install ./rns-1.0.1-py3-none-any.whl
```

На платформах, ограничивающих установку пользовательских пакетов через `pip` , возможно, потребуется вручную разрешить это, используя флаг командной строки `--break-system-packages` при установке. Это фактически не приведет к повреждению пакетов, если только вы не устанавливали Reticulum напрямую через менеджер пакетов вашей операционной системы.

```
pip install rns --break-system-packages
```

Для получения более подробных инструкций по установке, пожалуйста, обратитесь к разделу Примечания по установке для конкретной платформы.

После завершения установки может быть полезно обратиться к главе «Использование *Reticulum* в вашей системе».

2.1.1 Устранение проблем с зависимостями и установкой

На некоторых платформах могут отсутствовать бинарные пакеты для всех зависимостей, и установка `pip` может завершиться с сообщением об ошибке. В этих случаях проблему обычно можно решить, установив основные пакеты разработки для вашей платформы:

```
# Debian / Ubuntu / Derivatives
sudo apt install build-essential

# Arch / Manjaro / Derivatives
sudo pamac install base-devel

# Fedora
sudo dnf groupinstall "Development Tools" "Development Libraries"
```

После установки базовых пакетов разработки `pip` должен быть способен скомпилировать любые недостающие зависимости из исходного кода и завершить установку даже на платформах, где нет предварительно скомпилированных пакетов.

2.2 Попробуйте использовать программу на основе Reticulum

Если вы просто хотите попробовать использовать программу, построенную с помощью Reticulum, существует несколько различных программ, которые обеспечивают базовую связь и ряд других полезных функций, даже в сетях Reticulum с чрезвычайно низкой пропускной способностью.

Эти программы позволят вам получить представление о том, как работает Reticulum. Они были разработаны для эффективной работы в сетях на основе LoRa или пакетной радиосвязи, но также могут использоваться через высокоскоростные соединения, такие как локальная Сеть WiFi, проводной Ethernet, Интернет или любая их комбинация.

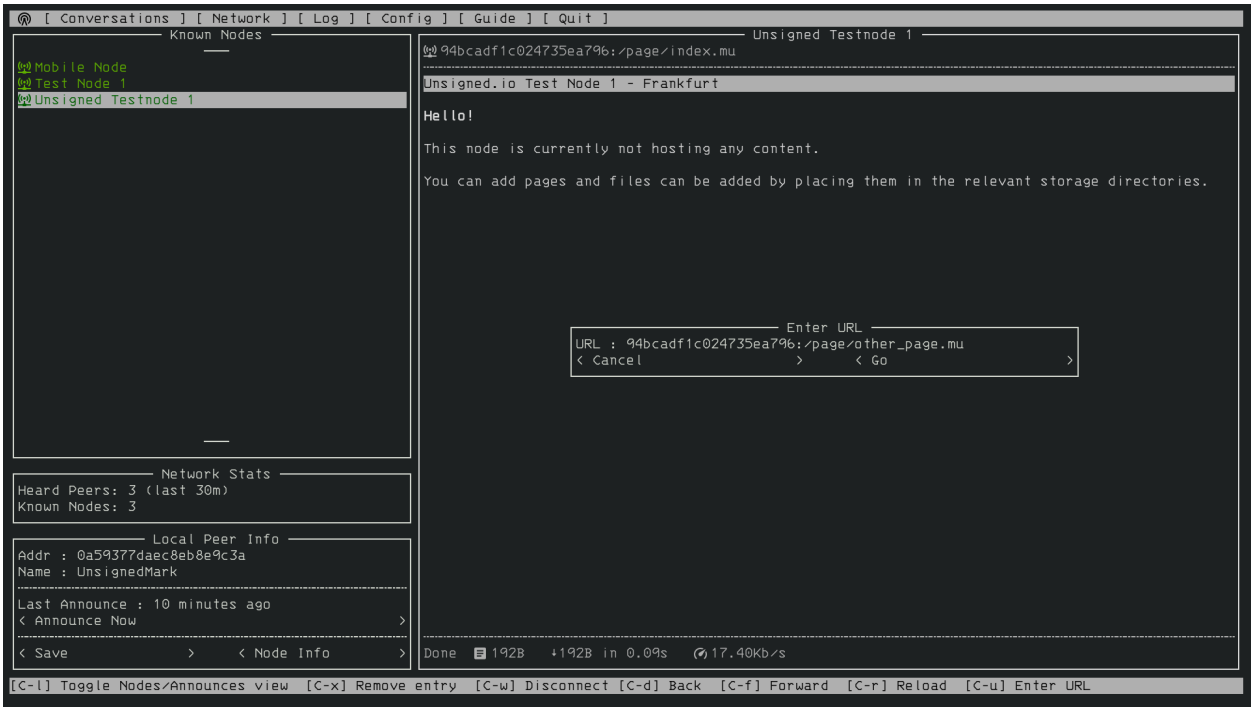
Таким образом, легко начать экспериментировать, не настраивая никаких радиоприемопередатчиков или инфраструктуры только для ознакомления. Запуска программ на отдельных устройствах, подключенных к одной и той же Сети WiFi, достаточно для начала, а физические радиоинтерфейсы могут быть добавлены позже.

2.2.1 Удалённая оболочка

Программа `rnsh` позволяет устанавливать полностью интерактивные сеансы удалённой оболочки через Reticulum. Она также позволяет передавать любую программу на удалённую систему или из неё, и аналогична тому, как работает `ssh`. `rnsh` очень эффективна и может обеспечивать полностью интерактивные сеансы оболочки даже по каналам с чрезвычайно низкой пропускной способностью, таким как LoRa или пакетная радиосвязь.

2.2.2 Сеть Nomad

Терминальная программа Nomad Network предоставляет полный набор зашифрованных коммуникаций, построенный с использованием Reticulum. Она включает зашифрованный обмен сообщениями (как прямой, так и с отложенной доставкой для пользователей офлайн), обмен файлами, а также имеет встроенный текстовый браузер и сервер страниц с поддержкой динамически отображаемых страниц, аутентификации пользователей и многого другого



Nomad Network — это клиентский интерфейс для протокола обмена сообщениями и информацией LXMF, еще одного проекта , созданного на базе Reticulum.

Вы можете установить Nomad Network через pip:

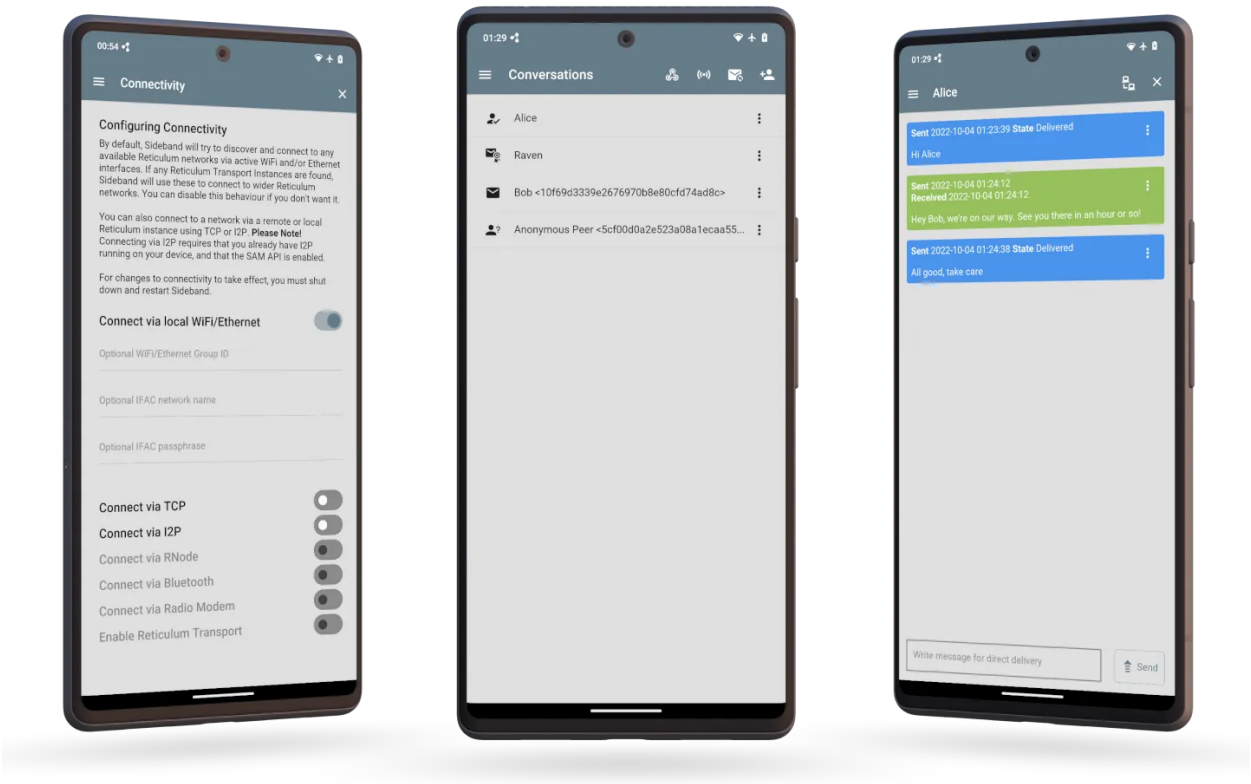
```
# Установить ...
pip install nomadnet

# ... и запустить
nomadnet
```

Примечание

Если вы впервые используете `pip` для установки программы в вашей системе, возможно, вам потребуется перезагрузить систему, чтобы программа стала доступной. Если при запуске программы вы получаете ошибку «команда не найдена» или аналогичную, перезагрузите систему и повторите попытку. В некоторых случаях вам может даже потребоваться вручную добавить путь установки `pip` в переменную среды `PATH` .

2.2.3 Sideband

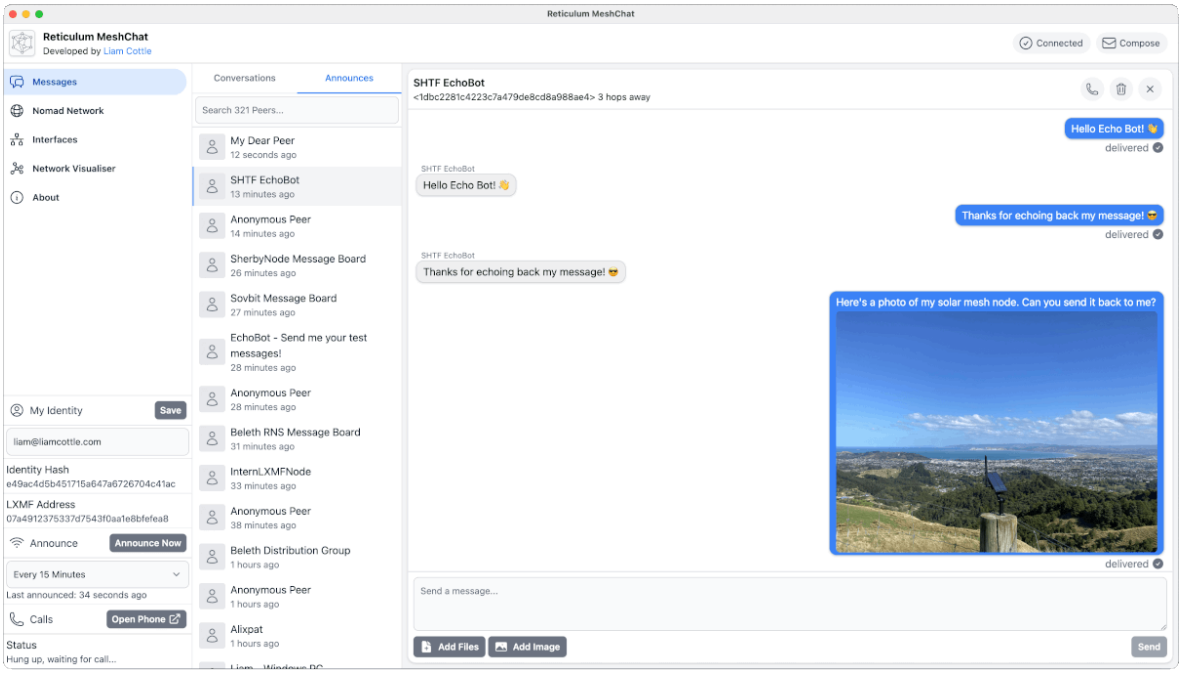
Если вы предпочитаете использовать программу с графическим пользовательским интерфейсом, вы можете обратить внимание на Sideband, который доступен для Android, Linux, macOS и Windows.



Sideband позволяет вам общаться с другими людьми или LXMF-совместимыми системами через сети Reticulum, используя LoRa, Packet Radio, WiFi, I2P, зашифрованные бумажные сообщения QR или любые другие поддерживаемые Reticulum средства. Он также взаимодействует с программой Nomad Network

2.2.4 MeshChat

Приложение Reticulum MeshChat — это удобный клиент LXMF для macOS и Windows, который также включает функцию голосовых вызовов и ряд других интересных функций.



Reticulum MeshChat, конечно же, также совместим с Sideband, Nomad Network или любым другим клиентом LXMF.

2.3 Использование встроенных утилит

Reticulum поставляется с набором встроенных утилит, которые облегчают управление вашей сетью, проверку подключения и делают Reticulum доступным для других программ в вашей системе.

Вы можете использовать `rnsd` для запуска Reticulum в качестве фоновой или приоритетной службы, а также утилиты `rnstatus`, `rnpath` и `rnprobe` для просмотра и запроса состояния и подключения к сети.

Чтобы узнать больше об этих служебных программах, ознакомьтесь с главой*Использование Reticulum в вашей системе* данного руководства.

2.4 Создание сети с Reticulum

Для создания сети вам потребуется указать один или несколько интерфейсов для использования Reticulum. Это делается в файле конфигурации Reticulum, который по умолчанию находится по адресу `~/.reticulum/config`. Пример файла конфигурации со всеми параметрами можно получить, используя команду `rnsd --exampleconfig`.

При первом запуске Reticulum создаст файл конфигурации по умолчанию с одним активным интерфейсом. Этот интерфейс по умолчанию использует ваши существующие сети Ethernet и Сеть WiFi (если таковые имеются) и позволяет вам общаться только с другими узлами Reticulum в пределах ваших локальных широковещательных доменов.

Для дальнейшей связи вам потребуется добавить один или несколько интерфейсов. Конфигурация по умолчанию включает ряд примеров: от использования TCP через интернет до интерфейсов LoRa и Packet Radio.

С Reticulum вам нужно лишь настроить, через какие интерфейсы вы хотите осуществлять связь. Нет необходимости конфигурировать адресные пространства, подсети, таблицы маршрутизации или другие параметры, к которым вы могли привыкнуть в других типах сетей.

Как только Reticulum узнает, какие интерфейсы он должен использовать, он автоматически обнаружит топологию и настроит передачу данных ко всем известным ему пунктам назначения.

В ситуациях, когда у вас уже есть установленная сеть Wi-Fi или Ethernet и множество устройств, которые хотят использовать те же внешние сетевые пути Reticulum (например, через LoRa), часто будет достаточно позволить одной системе действовать в качестве шлюза Reticulum, добавив любые внешние интерфейсы в конфигурацию этой системы, а затем включив на ней транспорт. Любое другое устройство в вашей локальной сети Wi-Fi сможет затем подключиться к этой более широкой сети Reticulum, просто используя конфигурацию по умолчанию (*AutoInterface*).

Возможно, примеров в файле конфигурации достаточно для начала работы. Если вам нужна дополнительная информация, вы можете прочитать главы *Building Networks* and *Interfaces* этого руководства.

2.5 Соединение экземпляров Reticulum через Интернет

В настоящее время Reticulum предлагает два интерфейса, подходящих для подключения экземпляров через Интернет: *TCP* and *I2P* . Каждый интерфейс предлагает различный набор функций, и пользователям Reticulum следует тщательно выбирать интерфейс, который наилучшим образом соответствует их потребностям.

TCPServerInterface позволяет пользователям размещать экземпляр, доступный по TCP/IP. Этот метод, как правило, быстрее, имеет меньшую задержку и более энергоэффективен, чем использование I2PInterface , однако он также раскрывает больше данных о хосте сервера.

TCP-соединения раскрывают IP-адрес как вашего экземпляра, так и сервера любому, кто может проверять соединение. Кто-то может использовать эту информацию для определения вашего местоположения или личности. Противники, инспектирующие ваши пакеты, могут записывать метаданные пакетов, такие как время передачи и размер пакета. Несмотря на то, что Reticulum шифрует трафик, TCP этого не делает, поэтому злоумышленник может использовать инспекцию пакетов, чтобы узнать, что система работает с Reticulum и какие другие IP-адреса к ней подключаются. Размещение общедоступного экземпляра через TCP также требует общедоступного IP-адреса, который большинство интернет-соединений больше не предлагают.

Интерфейс `I2PInterface` маршрутизирует сообщения через Невидимый Интернет-Протокол (I2P). Чтобы использовать этот интерфейс, пользователи также должны запускать демон I2P параллельно с `rnsd` . Для постоянно работающих узлов I2P рекомендуется использовать i2pd.

По умолчанию I2P будет шифровать и смешивать весь трафик, отправляемый через Интернет, а также скрывать IP-адреса отправителя и получателя экземпляра Reticulum. Запуск узла I2P также будет ретранслировать зашифрованные пакеты других пользователей I2P, что потребует дополнительной пропускной способности и вычислительной мощности, но также значительно затруднит атаки по времени и другие формы глубокой инспекции пакетов.

I2P также позволяет пользователям размещать глобально доступные экземпляры Reticulum с непубличных IP-адресов, а также за межсетевыми экранами и NAT.

В целом рекомендуется использовать узел I2P, если вы хотите разместить общедоступный экземпляр, сохраняя при этом анонимность. Если вас больше заботит производительность и немного более простая настройка, используйте TCP.

2.6 Подключение к общедоступной тестовой сети

Экспериментальная общедоступная тестовая сеть была сделана доступной добровольцами сообщества. Вы можете найти определения интерфейсов для добавления в ваш файл `.reticulum/config` на веб-сайте Reticulum или в Wiki сообщества.

Вы можете подключить свои устройства или экземпляры к одному или нескольким из них, чтобы получить доступ к любым сетям Reticulum, к которым они физически подключены. Просто добавьте один или несколько фрагментов интерфейса в файл конфигурации в разделе `[interface]` , как в примере ниже:

```
# TCP/IP interface to the BetweenTheBorders Hub (community-provided)
[[RNS Testnet BetweenTheBorders]]
  type = TCPClientInterface
  enabled = yes
  target_host = reticulum.betweentheborders.com
  target_port = 4242
```

Совет

В идеале настройте транспортный узел Reticulum, к которому ваши устройства смогут подключаться локально, а затем подключите этот транспортный узел к нескольким публичным точкам входа. Это обеспечит эффективное соединение и избыточность на случай отказа одного из них.

Многие другие экземпляры Reticulum подключаются к этой тестовой сети, и вы также можете присоединиться к ней через другие точки входа, если они вам известны. Абсолютно нет никакого контроля над топографией сети, ее использованием или типами подключаемых экземпляров. Она также будет периодически использоваться для тестирования различных сценариев сбоев, и нет никаких гарантий доступности или обслуживания. Ожидайте, что в этой сети будут происходить странные вещи, поскольку люди экспериментируют и пробуют новое.

Предупреждение

Вероятно, это само собой разумеется, но *не используйте точки входа тестовой сети в качестве жестко закодированных или стандартных интерфейсов в приложениях*, *которые вы поставляете пользователям* . При поставке приложений лучшей практикой является предоставление собственных решений для подключения по умолчанию, если это необходимо и применимо, или, в большинстве случаев, просто оставить пользователю выбор, к каким сетям подключаться и как.

2.7 Размещение публичных точек входа

Если вы хотите разместить публичную (или частную) точку входа в сеть Reticulum через Интернет, этот раздел предлагает несколько полезных советов. Вам понадобится физическая или виртуальная машина с публичным IP-адресом, доступная другим устройствам в Интернете.

Наиболее эффективный и производительный способ размещения подключаемой точки входа, поддерживающей множество пользователей, — использование `BackboneInterface` . Этот тип интерфейса полностью совместим с `TCPClientInterface` и `TCPServerInterface` типами, но намного быстрее и использует меньше системных ресурсов , позволяя вашему устройству обрабатывать тысячи подключений даже на небольших системах.

Также важно установить для вашего подключаемого интерфейса режим `gateway` , так как это значительно улучшит время сходимости сети и разрешение путей для всех, кто подключается к вашей точке входа.

```
# Этот пример демонстрирует магистральный интерфейс,
# настроенный для работы в качестве шлюза, позволяющего пользователям
# подключаться как к публичной, так и к частной сети.

[[Public Gateway]]
  type = BackboneInterface
  enabled = yes
  mode = gateway
  listen_on = 0.0.0.0
  port = 4242
```

Если вместо этого вы хотите создать частную точку входа из Интернета, вы можете использовать параметры *IFAC name* и *passphrase* для защиты вашего интерфейса с помощью имени сети и парольной фразы.

```
# Частная точка входа, требующая предварительно сообщенного #
имени сети и парольной фразы для подключения.

[[Private Gateway]]
  type = BackboneInterface
  enabled = yes
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  mode = gateway
  listen_on = 0.0.0.0
  port = 4242
  network_name = private_ret
  passphrase = 2owjajquafIanPecAc
```

Если вы размещаете точку входа в операционной системе, которая не поддерживает `BackboneInterface` , вы можете использовать `TCPServerInterface` , хотя он не будет таким производительным.

2.8 Добавление радиоинтерфейсов

После установки и запуска Reticulum вы можете добавить радиоинтерфейсы с использованием любого доступного совместимого оборудования. Reticulum поддерживает широкий спектр радиооборудования, и если у вас уже есть какое-либо из них, весьма вероятно, что оно будет работать с Reticulum. Информацию по настройке можно найти в разделе «Интерфейсы» данного руководства. Если у вас еще нет трансиверного оборудования, вы можете легко и недорого собрать *RNode* — универсальный дальнобойный цифровой радиоприемопередатчик, который легко интегрируется с Reticulum.

Для самостоятельной сборки требуется установить пользовательскую прошивку на поддерживаемую плату разработки LoRa с помощью скрипта автоматической установки. Пожалуйста, обратитесь к главе *«Оборудование связи»* за руководством. Если вы предпочитаете приобрести готовое устройство, вы можете обратиться к списку поставщиков. Для получения дополнительной информации о RNode вы также можете обратиться к следующим внешним ресурсам:

- Как сделать свои собственные RNodes
- Установка прошивки RNode на совместимые LoRa-устройства
- Приватный, безопасный и нецензурируемый обмен сообщениями через LoRa-сеть
- Прошивка RNode

Если у вас есть коммуникационное оборудование, которое ещё не поддерживается ни одним из *существующих типов интерфейсов* , но, по вашему мнению, подходит для использования с Reticulum, вы можете перейти на страницы обсуждений GitHub и предложить добавить интерфейс для данного оборудования.

2.9 Создание и использование пользовательских интерфейсов

Хотя Reticulum включает гибкий и широкий спектр встроенных интерфейсов, они не охватывают каждый мыслимый тип коммуникационного оборудования, которое Reticulum потенциально может использовать для связи.

Поэтому можно легко написать свои собственные модули интерфейса, которые могут быть загружены во время выполнения и использоваться наравне с любыми встроенными типами интерфейсов.

Для получения дополнительной информации по этой теме и примеров кода для разработки, пожалуйста, обратитесь к главе*Настройка интерфейсов* .

2.10 Разработка программы с Reticulum

Если вы хотите разрабатывать программы, использующие Reticulum, самый простой способ начать — это установить последнюю версию Reticulum через pip:

```
pip install rns
```

Вышеуказанная команда установит Reticulum и зависимости, и вы будете готовы импортировать и использовать RNS в своих программах. Следующим шагом, скорее всего, будет ознакомление с некоторыми *примерами программ* .

Весь API Reticulum задокументирован в главе *API Reference* этого руководства.

2.11 Участие в разработке Reticulum

Если вы хотите участвовать в разработке Reticulum и связанных с ним утилит, вам понадобится получить последнюю версию исходного кода с GitHub. В этом случае не используйте pip, а попробуйте следующий рецепт

```
# Install dependencies
pip install cryptography pyserial

# Clone repository
git clone https://github.com/markqvist/Reticulum.git

# Move into Reticulum folder and symlink library to examples folder
cd Reticulum
ln -s ../RNS ./Examples/

# Run an example
python Examples/Echo.py -s

# Unless you've manually created a config file, Reticulum will do so now,
# и немедленно выйти. Внесите необходимые изменения в файл:
nano ~/.reticulum/config

# ... и запустите пример снова.
python Examples/Echo.py -s

# Теперь вы можете повторить процесс на другом компьютере,
# и запустить тот же пример с -h, чтобы получить параметры командной строки.
python Examples/Echo.py -h

# Запустите пример в клиентском режиме, чтобы "пропинговать" сервер.
# Замените хеш ниже на фактический хеш назначения вашего сервера.
python Examples/Echo.py 174a64852a75682259ad8b921b8bf416

# Посмотрите другой пример
python Examples/Filetransfer.py -h
```

После того как вы поэкспериментировали с базовыми примерами, пришло время прочитать главу *«Понимание Reticulum»*. Перед отправкой первого запроса на извлечение, вероятно, будет хорошей идеей представиться на дискуссионном форуме GitHub или попросить одного из разработчиков или сопровождающих указать хорошее место для начала.

2.12 Примечания по установке для конкретной платформы

Некоторые платформы требуют немного иной процедуры установки или имеют различные особенности, о которых стоит знать. Они перечислены здесь.

2.12.1 Android

Reticulum можно использовать на Android различными способами. Самый простой способ начать — использовать приложение, например Sideband.

Для большего контроля и расширенных функций вы можете использовать Reticulum и связанные программы через приложение Termux, на момент написания доступное на F-droid.

Termux — это эмулятор терминала и среда Linux для устройств на базе Android, которая включает в себя возможность использования множества различных программ и библиотек, включая Reticulum.

Для использования Reticulum в среде Termux вам потребуется установить `python` и библиотеку `python-cryptography` с помощью pkg — менеджера пакетов, встроенного в Termux. После этого вы сможете использовать pip для установки Reticulum.

Внутри Termux выполните следующее:

```
# Сначала убедитесь, что индексы и пакеты обновлены.
pkg update
pkg upgrade

# Затем установите Python и библиотеку cryptography.
pkg install python python-cryptography

# Убедитесь, что pip обновлен, и установите модуль wheel.
pip install wheel pip --upgrade

# Установите Reticulum
pip install rns
```

Если по какой-то причине пакет `python-cryptography` недоступен для вашей платформы через менеджер пакетов Termux, вы можете попытаться собрать его локально на вашем устройстве, используя следующую команду:

```
# Сначала убедитесь, что индексы и пакеты обновлены.
pkg update
pkg upgrade

# Затем установите зависимости для библиотеки cryptography.
pkg install python build-essential openssl libffi rust

# Убедитесь, что pip обновлен, и установите модуль wheel.
pip install wheel pip --upgrade

# Чтобы установщик мог собрать модуль cryptography, # нам нужно
сообщить ему, для какой платформы мы компилируем:
export CARGO_BUILD_TARGET="aarch64-linux-android"

# Запустите процесс установки для модуля cryptography.
# В зависимости от вашего устройства, это может занять несколько минут, #
так как модуль должен быть скомпилирован локально на вашем устройстве.
pip install cryptography

# Если вышеуказанная установка прошла успешно, вы теперь можете
установить # Reticulum и любое связанное программное обеспечение
pip install rns
```

Также возможно включить Reticulum в приложения, скомпилированные и распространяемые как Android APK. Подробное руководство и пример исходного кода будут включены здесь позднее. До тех пор вы можете использовать исходный код Sideband в качестве примера и отправной точки.

2.12.2 ARM64

На некоторых архитектурах, включая ARM64, не все зависимости имеют предварительно скомпилированные бинарные файлы. На таких системах вам может потребоваться установить `python3-dev` (или аналогичный пакет) перед установкой Reticulum или программ, которые зависят от Reticulum.

```
# Install Python and development packages
sudo apt update
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
sudo apt install python3 python3-pip python3-dev

# Install Reticulum
python3 -m pip install rns
```

При установке этих пакетов pipсможет локально собрать любые отсутствующие зависимости в вашей системе.

2.12.3 Debian Bookworm

В версиях Debian, выпущенных после апреля 2023 года, по умолчанию больше невозможно использовать `pip` для установки пакетов в вашу систему. К сожалению, вместо этого вам придется использовать команду `pipx`, которая помещает установленные пакеты в изолированную среду. Это не должно негативно повлиять на Reticulum, но не будет работать для включения и использования Reticulum в ваших собственных скриптах и программах.

```
# Install pipx
sudo apt install pipx

# Make installed programs available on the command line
pipx ensurepath

# Install Reticulum
pipx install rns
```

В качестве альтернативы, вы можете восстановить нормальное поведение `pip`, создав или отредактировав файл конфигурации, расположенный по адресу `~/.config/pip/pip.conf`, и добавив следующий раздел:

```
[global]
break-system-packages = true
```

Для однократной установки Reticulum, без глобального включения опции `break-system-packages`, вы можете использовать следующую команду:

```
pip install rns --break-system-packages
```

Примечание

Директива `--break-system-packages` — это несколько вводящий в заблуждение выбор слов. Ее установка, конечно, не повредит никакие системные пакеты, а просто позволит устанавливать `pip` пакеты для пользователя и всей системы. Хотя это может в редких случаях привести к конфликтам версий, это, как правило, не создает никаких проблем, особенно в случае установки Reticulum.

2.12.4 MacOS

Для установки Reticulum на macOS вам потребуется установить Python и менеджер пакетов `pip`.

Системы под управлением macOS могут сильно различаться по наличию предустановленного Python, его версии, а также по тому, установлен ли и настроен ли менеджер пакетов `pip`. Если сомневаетесь, вы можете загрузить и установить Python вручную.

Когда Python и `pip` доступны в вашей системе, просто откройте окно терминала и используйте одну из следующих команд:

```
# Установите Reticulum и утилиты с помощью pip:
pip3 install rns

# В некоторых версиях может потребоваться использовать
флаг --break-system-packages для установки:
pip3 install rns --break-system-packages
```

Примечание

Директива `--break-system-packages` — это несколько вводящий в заблуждение выбор слов. Ее установка, конечно, не повредит никакие системные пакеты, а просто позволит устанавливать `pip` пакеты для пользователя и всей системы. Хотя это может в редких случаях привести к конфликтам версий, это, как правило, не создает никаких проблем, особенно в случае установки Reticulum.

Кроме того, некоторые комбинации версий macOS и Python требуют ручного добавления каталога установленных пакетов `pip` в переменную среды *PATH*, прежде чем вы сможете использовать установленные команды в своем терминале. Обычно достаточно добавить следующую строку в ваш скрипт инициализации оболочки (например, `~/.zshrc`):

```
export PATH=$PATH:~/Library/Python/3.9/bin
```

Отрегулируйте версию Python и расположение скрипта инициализации оболочки в соответствии с вашей системой.

2.12.5 OpenWRT

В системах OpenWRT с достаточным объемом хранилища и памяти вы можете установить Reticulum и связанные утилиты, используя менеджер пакетов *opkg* и pip .

Примечание

На момент выпуска этого руководства ведется работа по созданию предварительно собранных пакетов Reticulum для OpenWRT с полной конфигурацией, обслуживанием и `uci` интеграцией. Пожалуйста, см. репозитории feed-reticulum и reticulum-openwrt для получения дополнительной информации.

Для установки Reticulum на OpenWRT сначала войдите в сессию командной строки, а затем используйте следующие инструкции:

```
# Установить зависимости
opkg install python3 python3-pip python3-cryptography python3-pyserial

# Установите Reticulum
pip install rns

# Запустить rnsd с включенным отладочным
логированием rnsd -vvv
```

Примечание

Вышеуказанные инструкции были проверены и протестированы только на OpenWRT 21.02. Вполне вероятно, что другие версии могут потребовать слегка измененных команд установки или названий пакетов. Вам также понадобится достаточно свободного места в вашей overlay FS и достаточно свободной оперативной памяти для фактического запуска Reticulum и любых связанных программ и утилит.

В зависимости от конфигурации вашего устройства, вам может потребоваться настроить правила брандмауэра для обеспечения работы подключения Reticulum к вашему устройству и от него. Пока надлежащая упаковка не будет готова, вам также потребуется вручную создать службу или скрипт автозапуска для автоматического запуска Reticulum при загрузке.

Обратите также внимание, что *AutoInterface* требует включения локальных IPv6-адресов для любых устройств Ethernet и Wi-Fi, которые вы собираетесь использовать. Если `ip a` показывает адрес, начинающийся с `fe80::` для данного устройства, то `AutoInterface` должно работать для этого устройства.

2.12.6 Raspberry Pi

В настоящее время рекомендуется использовать 64-битную версию Raspberry Pi OS, если вы хотите запустить Reticulum на компьютерах Raspberry Pi, поскольку для 32-битных версий не всегда доступны пакеты для некоторых зависимостей. Если Python и менеджер пакетов *pip* ещё не установлены, сделайте это сначала, а затем установите Reticulum с помощью *pip* .

```
# Установить зависимости
sudo apt install python3 python3-pip python3-cryptography python3-pyserial

# Install Reticulum
pip install rns --break-system-packages
```

Примечание

Директива `--break-system-packages` — это несколько вводящий в заблуждение выбор слов. Ее установка, конечно, не повредит никакие системные пакеты, а просто позволит устанавливать `pip` пакеты для пользователя и всей системы. Хотя это может в редких случаях привести к конфликтам версий, это, как правило , не создает никаких проблем, особенно в случае установки Reticulum.

Хотя Reticulum можно установить и запустить на 32-битных ОС Raspberry Pi, это потребует ручной настройки и установки необходимых зависимостей для сборки, и в данном руководстве это подробно не описано.

2.12.7 RISC-V

На некоторых архитектурах, включая RISC-V, не все зависимости имеют предварительно скомпилированные бинарные файлы. На таких системах вам может потребоваться установить `python3-dev` (или аналогичный пакет) перед установкой Reticulum или программ, которые зависят от Reticulum.

```
# Install Python and development packages
sudo apt update
sudo apt install python3 python3-pip python3-dev

# Install Reticulum
python3 -m pip install rns
```

При установке этих пакетов `pip`сможет локально собрать любые отсутствующие зависимости в вашей системе.

2.12.8 Ubuntu Lunar

В версиях Ubuntu, выпущенных после апреля 2023 года, по умолчанию больше невозможно использовать `pip` для установки пакетов в вашу систему. К сожалению, вместо этого вам придется использовать команду `pipx` , которая помещает установленные пакеты в изолированную среду. Это не должно негативно повлиять на Reticulum, но не будет работать для включения и использования Reticulum в ваших собственных скриптах и программах.

```
# Install pipx
sudo apt install pipx
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# Make installed programs available on the command line
pipx ensurepath

# Install Reticulum
pipx install rns
```

В качестве альтернативы, вы можете восстановить нормальное поведение `pip`, создав или отредактировав файл конфигурации, расположенный по адресу `~/.config/pip/pip.conf` , и добавив следующий раздел:

```
[global]
break-system-packages = true
```

Для однократной установки Reticulum, без глобального включения опции `break-system-packages` , вы можете использовать следующую команду:

```
pip install rns --break-system-packages
```

Примечание

Директива `--break-system-packages` — это несколько вводящий в заблуждение выбор слов. Ее установка, конечно, не повредит никакие системные пакеты, а просто позволит устанавливать `pip` пакеты для пользователя и всей системы. Хотя это может в редких случаях привести к конфликтам версий, это, как правило, не создает никаких проблем, особенно в случае установки Reticulum.

2.12.9 Windows

В операционных системах Windows самый простой способ установить Reticulum — это использовать менеджер пакетов `pip` из командной строки (либо командную строку, либо Windows Powershell).

Если у вас ещё не установлен Python, скачайте и установите Python. На момент публикации данного руководства рекомендуемая версия — Python 3.12.7.

Важно! Когда установщик запросит, обязательно добавьте программу Python в переменные среды PATH. Если вы этого не сделаете, вы не сможете использовать установщик `pip` или запускать включённые утилиты Reticulum (такие как `rnsd` и `rnstatus`) из командной строки.

После установки Python откройте командную строку или Windows Powershell и введите:

```
pip install rns
```

Теперь вы можете использовать Reticulum и все включённые утилиты непосредственно из вашего предпочтительного интерфейса командной строки.

2.13 Reticulum на чистом Python

Предупреждение

Если вы используете пакет `rnspure` для запуска Reticulum на системах, которые не поддерживают PyCA/cryptography, важно, чтобы вы прочитали и поняли раздел *Криптографические примитивы* данного руководства

В некоторых редких случаях и на более необычных типах систем невозможно установить одну или несколько зависимостей . В таких ситуациях вы можете использовать пакет `rnspure` вместо пакета `rns` или использовать `pip` с `--no-dependencies`

опция командной строки. Пакет `rnspure` не требует внешних зависимостей для установки. Обратите внимание, что фактическое содержимое пакетов `rns` и `rnspure` абсолютно идентично . *Единственное отличие состоит в том, что пакет* `rnspure` не указывает зависимостей, необходимых для установки.

Независимо от способа установки и запуска Reticulum, он будет загружать внешние зависимости только в том случае, если они необходимы и доступны . Если, например, вы хотите использовать Reticulum в системе, которая не поддерживает `pyserial` , это вполне возможно сделать с помощью пакета *rnspure* , но Reticulum не сможет использовать последовательные интерфейсы. Все остальные доступные модули будут по-прежнему загружаться по мере необходимости.

ГЛАВА

ТРИ

ИСПОЛЬЗОВАНИЕ RETICULUM В ВАШЕЙ СИСТЕМЕ

Reticulum не устанавливается как драйвер или модуль ядра, как можно было бы ожидать от сетевого стека. Вместо этого Reticulum распространяется как модуль Python, содержащий сетевое ядро, а также набор служебных и демонических программ

Это означает, что для его установки или использования не требуются особые привилегии. Он также очень легковесен , его легко переносить и устанавливать на новые системы.

Когда Reticulum установлен, любая программа или приложение, использующее Reticulum, автоматически загрузит и инициализирует Reticulum при запуске, если он еще не запущен.

Во многих случаях этого подхода достаточно. Когда любая программа должна использовать Reticulum, он загружается, инициализируется, поднимаются интерфейсы, и программа теперь может обмениваться данными по любым доступным сетям Reticulum. Если другая программа запускается и также хочет получить доступ к той же сети Reticulum, уже запущенный экземпляр просто используется совместно. Это работает для любого количества программ, запущенных одновременно, и очень просто в использовании, но в зависимости от вашего варианта использования существуют и другие варианты.

3.1 Конфигурация и данные

Reticulum хранит всю необходимую для работы информацию в одном каталоге файловой системы. При запуске Reticulum будет искать действительный каталог конфигурации в следующих местах:

- `/etc/reticulum`
- `~/.config/reticulum`
- `~/.reticulum`

Если существующий каталог конфигурации не найден, будет создан каталог `~/.reticulum` , и здесь будет автоматически создана конфигурация по умолчанию. При желании вы можете переместить его в одно из других мест.

Также возможно использовать совершенно произвольные каталоги конфигурации, указывая соответствующие параметры командной строки при запуске программ на основе Reticulum. Вы также можете запускать несколько отдельных экземпляров Reticulum на одной физической системе, либо изолированно друг от друга, либо соединенных между собой.

В большинстве случаев одной физической системе потребуется запустить только один экземпляр Reticulum. Он может быть запущен при загрузке как системная служба или просто активирован, когда программа в нем нуждается. В любом случае, любое количество программ, работающих на одной системе, будет автоматически использовать один и тот же экземпляр Reticulum, если конфигурация это позволяет, что является поведением по умолчанию.

Вся конфигурация Reticulum находится в файле `~/.reticulum/config`. При первом запуске Reticulum в новой системе создается базовый, но полностью функциональный файл конфигурации. Конфигурация по умолчанию выглядит так:

```
# Это файл конфигурации Reticulum по умолчанию.
# Вам, вероятно, следует отредактировать его, чтобы включить любые дополнительные # интер-
фейсы и настройки, которые могут вам понадобиться.
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# Только самые основные параметры включены в эту конфигурацию по
умолчанию. Чтобы увидеть более подробный и гораздо более длинный #
 пример конфигурации, вы можете выполнить команду:
# rnsd --exampleconfig


[reticulum]

# Если вы включите Transport, ваша система будет маршрутизировать трафик
# для других узлов, объявляет маршруты и обрабатывает запросы путей.
# Это следует делать для систем, которые подходят для ра-
боты в качестве транспортных узлов, то есть если они
стационарны и всегда включены. Эта директива необяза-
тельна и может быть удалена # для краткости.

enable_transport = No


# По умолчанию первая программа, запускающая сетевой стек
Reticulum, создаст общий экземпляр, с которым смогут взаимодей-
ствовать другие программы. Только общий экземпляр # открывает
все сконфигурированные интерфейсы напрямую, а другие # локаль-
ные программы взаимодействуют с общим экземпляром через # ло-
кальный сокет. Это полностью прозрачно для # пользователя и,
как правило, должно быть включено. Эта директива # необяза-
тельна и может быть удалена для краткости.

share_instance = Yes


# Если вы хотите запустить несколько *различных* общих экземпляров #
 на одной системе, вам потребуется указать разные # имена экземпляров
для каждого. На платформах, поддерживающих доменные # сокеты,
это можно сделать с помощью опции instance_name:

instance_name = default

# Некоторые платформы не поддерживают доменные сокеты,
и в этом # случае вы можете изолировать различные экземпляры
, # указав для каждого уникальный набор портов:

# shared_instance_port = 37428
# instance_control_port = 37429


# Если вы хотите явно использовать TCP для связи общих
экземпляров # вместо доменных сокетов, это также # воз-
можно, используя следующую опцию:

# shared_instance_type = tcp
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# В системах, где запущенные экземпляры могут не иметь доступа #
 к одному и тому же общему каталогу конфигурации Reticulum, #
все еще возможно обеспечить полную интерактивность для # запу-
щенных экземпляров, вручную указав общий ключ RPC. # Почти во
всех случаях эта опция не требуется, но # она может быть полезна
на таких операционных системах, как Android.
# Ключ должен быть указан в виде байтов в шестнадцатеричном формате.

# rpc_key = e5c032d3ec4e64a6aca9927ba8ab73336780f6d71790


# Возможно разрешить удаленное управление системами
Reticulum # с использованием различных встроенных утилит,
таких как # rnstatus и rnpath. Вам нужно будет указать один
или # несколько хешей Reticulum Identity для аутентификации #
 запросов от клиентских программ. Для этой цели вы можете
# использовать существующие файлы идентификации или гене-
рировать новые с помощью # утилиты rnid.

# enable_remote_management = yes
# remote_management_allowed = 9fb6d773498fb3feda407ed8ef2c3229,
↪2d882c5586e548d79b5af27bca1776dc


# Вы можете настроить Reticulum на панику и принудительное закрытие
# в случае возникновения неисправимой ошибки интерфейса, например, #
исчезновения аппаратного устройства для интерфейса. Это # необяза-
тельная директива, и её можно опустить для краткости.
# Это поведение отключено по умолчанию.

# panic_on_interface_error = No


# Когда Transport включен, можно разрешить # Экземпляру транс-
портного протокола отвечать на запросы зондирования от #
утилиты rnprobe. Это может быть полезным инструментом
для проверки # подключения. Когда эта опция включена, цель зон-
дирования # будет сгенерирована из Identity # Экземпляра транс-
портного протокола и выведена в лог при запуске.
# Опционально и отключено по умолчанию.

# respond_to_probes = No


[logging]
# Допустимые уровни логирования от 0 до 7:
#   0: Логировать только критическую информацию
#   1: Логировать ошибки и более низкие уровни логирования
#   2: Логировать предупреждения и более низкие уровни логирования
#   3: Логировать уведомления и более низкие уровни логирования
#   4: Логировать информацию и более низкие уровни (это значение по умолчанию)
#   5: Подробное логирование
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
#    6: Отладочное логирование
#    7: Экстремальное логирование

loglevel = 4


# Раздел interfaces определяет физические и виртуальные # интер-
фейсы, которые Reticulum будет использовать для связи. Этот # раз-
дел содержит примеры различных типов интерфейсов. Вы можете
изменить их или использовать их в качестве основы для # собственной
конфигурации, или просто удалить неиспользуемые.

[interfaces]

  # Этот интерфейс обеспечивает связь с другими # локальными уз-
  лами Reticulum по UDP. Ему не # требуется функциональная IP-ин-
  фраструктура, например маршрутизаторы # или DHCP-серверы,
  но потребуется, чтобы в вашей операционной системе # был
  включен как минимум локальный IPv6, что # должно быть вклю-
  чено по умолчанию почти в любой ОС. Дополнительные параметры
  конфигурации # см. в Руководстве Reticulum.

  [[Default Interface]]
    type = AutoInterface
    interface_enabled = True
```

Если инфраструктура Reticulum уже существует локально, вам, вероятно, не нужно ничего менять, и вы, возможно, уже подключены к более широкой сети. В противном случае, вам, вероятно, потребуется добавить соответствующие интерфейсы в конфигурацию, чтобы обмениваться данными с другими системами.

Вы можете сгенерировать гораздо более подробный пример конфигурации, выполнив команду:

```
rnsd --exampleconfig
```

Вывод включает примеры для большинства типов интерфейсов, поддерживаемых Reticulum, наряду с дополнительными опциями и параметрами конфигурации.

Рекомендуется прочитать комментарии и пояснения в вышеуказанной конфигурации по умолчанию. Это научит вас основным понятиям, которые необходимо понять для настройки вашей сети. После этого ознакомьтесь с главой Интерфейсы данного руководства.

3.2 Включенные утилиты

Reticulum включает ряд полезных утилит как для управления вашими сетями Reticulum, так и для выполнения общих задач по сетям Reticulum, таких как передача файлов на удаленные системы и удаленное выполнение команд и программ.

Если вы часто используете Reticulum из нескольких разных программ или просто хотите, чтобы Reticulum оставался доступным постоянно, например, если вы размещаете транспортный узел, вы можете запустить Reticulum как отдельную службу, которую могут использовать другие программы, приложения и службы.

3.2.1 Утилита rnsd

Запустить Reticulum как службу очень просто. Просто запустите включенную команду `rnsd`. Когда `rnsd` запущена, она будет держать все настроенные интерфейсы открытыми, обрабатывать транспорт, если он включён, и позволять любым другим программам немедленно использовать сеть Reticulum, для которой она настроена.

Вы даже можете запустить несколько экземпляров `rnsd`с различными конфигурациями на одной и той же системе.

Примеры использования

Запуск `rnsd`:

```
$ rnsd

[2023-08-18 17:59:56] [Notice] Started rnsd version 0.5.8
```

Запуск `rnsd`в режиме службы, обеспечивающий отправку всего вывода журнала непосредственно в файл:

```
$ rnsd -s
```

Создать подробный и детальный пример конфигурации с объяснениями всех различных параметров конфигурации, а также примеры конфигурации интерфейса:

```
$ rnsd --exampleconfig
```

Все параметры командной строки

```
использование: rnsd.py [-h] [--config CONFIG] [-v] [-q] [-s] [--exampleconfig] [--version]

Демон Сетевого стека Reticulum

опции:
  -h, --help         показать это справочное сообщение и выйти
  --config CONFIG    путь к альтернативному каталогу конфигурации Reticulum
  -v, --verbose
  -q, --quiet
  -s, --service      rnsd работает как служба и должен записывать логи в файл
  -i, --interactive drop into interactive shell after initialisation --
  exampleconfig вывести подробный пример конфигурации в stdout и выйти --
  version            показать номер версии программы ' и выйти
```

Вы можете легко добавить `rnsd` в качестве постоянно работающей службы, настроив службу .

3.2.2 Утилита rnstatus

Используя утилиту `rnstatus`, вы можете просматривать состояние настроенных интерфейсов Reticulum, подобно программе `ifconfig`.

Примеры использования

Запуск `rnstatus`:

```
$ rnstatus

Общий экземпляр[37428]
    Статус : Активен
    Обслуживает : 1 программа
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
   Скорость : 1.00 Гбит/с
    Трафик : 83.13 КБ↑
             86.10 КБ↓

Автоинтерфейс[Локальный]
   Статус : Активен
   Режим : Полный
   Скорость : 10.00 Мбит/с
   Пиры : 1 доступен
    Трафик : 63.23 КБ↑
             80.17 КБ↓

TCPИнтерфейс[RNS Testnet Dublin/dublin.connect.reticulum.network:4965]
   Статус : Активен
   Режим : Полный
   Скорость : 10.00 Мбит/с
    Трафик : 187.27 КБ↑
             74.17 КБ↓

RNodeИнтерфейс[RNode UHF]
   Статус : Активен
   Режим : Точка доступа
   Скорость: 1.30 кбит/с
   Доступ: 64-бит IFAC по <...e702c42ba8>
    Трафик: 8.49 КБ↑
             9.23 КБ↓

Экземпляр транспортного протокола Reticulum <5245a8efe1788c6a1cd36144a270e13b> запущен
```

Фильтровать вывод для отображения только некоторых интерфейсов:

```
$ rnstatus rnode

RNodeИнтерфейс[RNode UHF]
   Статус : Активен
   Режим : Точка доступа
   Скорость: 1.30 кбит/с
   Доступ: 64-бит IFAC по <...e702c42ba8>
   Трафик: 8.49 КБ↑
            9.23 КБ↓

Экземпляр транспортного протокола Reticulum <5245a8efe1788c6a1cd36144a270e13b> запущен
```

Все параметры командной строки

```
использование: rnstatus [-h] [--config CONFIG] [--version] [-a
        ] [-A] [-l] [-s SORT] [-r] [-j] [-R hash] [-i path]
                [-w seconds] [-v] [filter]

Статус Сетевого стека Reticulum

позиционные аргументы:
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  фильтр                отображать только интерфейсы, имена которых включают фильтр

опции:
  -h, --help            показать это справочное сообщение и выйти
  --config CONFIG       путь к альтернативному каталогу конфигурации Reticulum
  --version             показать номер версии программы' и выйти
  -a, --all             показать все интерфейсы
  -A, --announce-stats показать статистику объявлений
  -l, --link-stats      показать статистику канала
  -s SORT, --sort SORT сортировать интерфейсы по [rate, traffic, rx, tx, announces, arx, atx,
↪held]
  -r, --reverse         обратная сортировка
  -j, --json            вывод в формате JSON
  -R hash               хеш идентификатора транспортного протокола удаленного экземпляра для получения статуса от
↪(требуется -i)
  -i path               путь к идентификатору, используемому для удаленного управления
  -w seconds            тайм-аут до прекращения удаленных запросов
  -v, --verbose
```

Примечание

При использовании `-R` для запроса удаленного экземпляра транспортного протокола вы также должны указать `-i` с путём к файлу идентификатора управления, который авторизован для удаленного управления на целевой системе.

3.2.3 Утилита rnid

С помощью утилиты `rnid` вы можете генерировать, управлять и просматривать идентификаторы Reticulum. Программа также может вычислять хеши адресатов и выполнять шифрование и дешифрование файлов.

Используя `rnid` , можно асимметрично шифровать файлы и информацию для любого целевого хеша Reticulum, а также создавать и проверять криптографические подписи.

Примеры использования

Создать новую идентификацию:

```
$ rnid -g ./new_identity
```

Показать информацию о ключе идентификации:

```
$ rnid -i ./new_identity -p

Загруженная идентификация <984b74a3f768bef236af4371e6f248cd> из new_id
Открытый ключ : 0f4259fef4521ab75a3409e353fe9073eb10783b4912a6a9937c57bf44a62
c1e Закрытый ключ : Скрыт
```

Зашифровать файл для пользователя LXMF:

```
$ rnid -i 8dd57a7382268096460893335a6b03695 -e my_file.txt

Идентификация <bc7291552be7a58f3615229904651165c> для назначения
↪<8dd57a7382268096460893335a6b03695>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
Шифрование my_file.txt
Файл my_file.txt зашифрован для <bc7291552be7a58f361522990465165c> в my_file.txt.rfe
```

Если идентификатор адресата еще не известен, вы можете получить его из сети, используя параметр командной строки `-R`:

```
$ rnid -R -i 30602def3b3506a28ed33db6f60cc6c9 -e my_file.txt

Запрос неизвестного идентификатора для <30602def3b3506a28ed33db6f60cc6c9>...
Получен идентификатор <2b489d06eaf7c543808c76a5332a447d> для адресата
↪<30602def3b3506a28ed33db6f60cc6c9> из сети
Шифрование my_file.txt
Файл my_file.txt зашифрован для <2b489d06eaf7c543808c76a5332a447d> в my_file.txt.rfe
```

Расшифровать файл, используя идентификатор Reticulum, для которого он был зашифрован:

```
$ rnid -i ./my_identity -d my_file.txt.rfe

Загружен идентификатор <2225fdeecaf6e2db4556c3c2d7637294> из ./my_identity
Расшифровка ./my_file.txt.rfe...
Файл ./my_file.txt.rfe расшифрован с <2225fdeecaf6e2db4556c3c2d7637294> в ./my_file.txt
```

Все параметры командной строки

```
Использование: rnid.py [-h] [--config path] [-i identity] [-g path] [-v] [-q] [-a
               aspects] [-H aspects] [-e path] [-d path] [-s path] [-V path] [-r path] [-w path
               ] [-f] [-R] [-t seconds] [-p] [-P] [--version]

Утилита идентификации и шифрования Reticulum

опции:
  -h, --help            показать это справочное сообщение и выйти
  --config path         Путь к альтернативному каталогу конфигурации Reticulum
  -i, --identity identity
                        шестнадцатеричный идентификатор Reticulum или хеш назначения, или путь к
↪Файл идентификации
  -g, --generate file   сгенерировать новую идентификацию
  -m, --import identity_data
                        импортировать идентификацию Reticulum в шестнадцатеричном формате, base32 или base64
  -x, --export          экспортировать идентификацию в шестнадцатеричный формат, base32 или base64
  -v, --verbose         увеличить детализацию
  -q, --quiet         уменьшить детализацию
  -a, --announce aspects
                        анонсировать назначение на основе этой идентификации -H, --
  hash aspects показать хеши назначения для других аспектов этой идентификации
  -e, --encrypt file    зашифровать файл
  -d, --decrypt file    расшифровать файл
  -s, --sign path       подписать файл
  -V, --validate path   проверить подпись
  -r, --read file       путь к входному файлу
  -w, --write file      путь к выходному файлу
  -f, --force           записывать вывод, даже если он перезаписывает существующие файлы
  -R, --request         запрашивать неизвестные идентификаторы из сети
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  -t seconds            тайм-аут запроса идентификатора перед отказом
  -p, --print-identity вывести информацию об идентификаторе и выйти
  -P, --print-private   разрешить отображение закрытых ключей
  -b, --base64          Использовать ввод и вывод в кодировке base64
  -B, --base32          Использовать ввод и вывод в кодировке base32
  --version             показать номер версии программы' и выйти
```

3.2.4 Утилита rnpath

С помощью утилиты `rnpath` вы можете искать и просматривать пути для адресатов в сети Reticulum.

Примеры использования

Определить путь к адресату:

```
$ rnpath c89b4da064bf66d280f0e4d8abfd9806

Путь найден, адресат <c89b4da064bf66d280f0e4d8abfd9806> находится в 4 переходах через
↪<f53a1c4278e0726bb73fcc623d6ce763> на TCPInterface[Testnet/dublin.connect.reticulum.
↪сеть:4965]
```

Все параметры командной строки

```
использование: rnpath [-h] [--config CONFIG] [--version] [-t] [-m
               hops] [-r] [-d] [-D] [-x] [-w seconds] [-R hash] [-i path]
                [-W seconds] [-j] [-v] [destination]

Утилита обнаружения пути Reticulum

позиционные аргументы:
  назначение            шестнадцатеричный хеш назначения

опции:
  -h, --help            показать это справочное сообщение и выйти
  --config CONFIG       путь к альтернативному каталогу конфигурации Reticulum
  --version             показать номер версии программы' и выйти
  -t, --table           показать все известные пути
  -m hops, --max hops   максимальное количество прыжков для фильтрации таблицы путей
  -r, --rates           показать информацию о скорости объявлений
  -d, --drop            удалить путь к назначению
  -D, --drop-announces удалить все поставленные в очередь объявления
  -x, --drop-via        удалить все пути через указанный Экземпляр транспортного протокола
  -w seconds            таймаут до отказа
  -R hash               хеш идентификатора транспорта удаленного экземпляра для управления
  -i путь               путь к идентификатору, используемому для удаленного управления
  -W секунд             тайм-аут до прекращения удаленных запросов
  -j, --json            вывод в формате JSON
  -v, --verbose
```

3.2.5 Утилита rnprobe

Утилита `rnprobe` позволяет проверять доступность целевого объекта, аналогично программе `ping`. Обратите внимание, что на запросы будут отвечать только в том случае, если указанный целевой объект настроен на отправку подтверждений для полученных пакетов. У многих целевых объектов эта опция не будет включена, поэтому большинство целевых объектов не будут доступны для проверки.

Вы можете включить целевой объект ответа на запросы в Экземплярах транспортного протокола Reticulum, установив директиву конфигурации `respond_to_probes`. Reticulum затем выведет целевой объект запроса в журнал при запуске Экземпляра транспортного протокола.

Примеры использования

Проверка целевого объекта:

```
$ rnprobe rnstransport.probe 2d03725b327348980d570f739a3a5708

Отправлен 16-байтовый запрос к <2d03725b327348980d570f739a3a5708>
Получен действительный ответ от <2d03725b327348980d570f739a3a5708>
Время кругового пути составляет 38.469 миллисекунд через 2 перехода
```

Отправить более крупный зонд:

```
$ rnprobe rnstransport.probe 2d03725b327348980d570f739a3a5708 -s 256

Отправлен 16-байтовый зонд на <2d03725b327348980d570f739a3a5708>
Получен действительный ответ от <2d03725b327348980d570f739a3a5708>
Время кругового пути составляет 38,781 миллисекунды за 2 перехода
```

Если интерфейс, получающий ответы зонда, поддерживает передачу радиопараметров, таких как **RSSI** и **SNR**, утилита `rnprobe` также выведет их как часть результата.

```
$ rnprobe rnstransport.probe e7536ee90bd4a440e130490b87a25124

Отправлен 16-байтовый зонд на <e7536ee90bd4a440e130490b87a25124>
Получен действительный ответ от <e7536ee90bd4a440e130490b87a25124>
Время кругового пути составляет 1,809 секунды за 1 переход [RSSI -73 dBm] [SNR 12.0 dB]
```

Все параметры командной строки

```
использование: rnprobe [-h] [--config CONFIG] [-s SIZE] [-n
               PROBES] [-t seconds] [-w seconds] [--version] [-v
               ] [full_name] [destination_hash]

Утилита зондирования Reticulum

позиционные аргументы:
  full_name             полное имя назначения в точечной нотации
  destination_hash      шестнадцатеричный хеш назначения

опции:
  -h, --help            показать это справочное сообщение и выйти
  --config CONFIG       путь к альтернативному каталогу конфигурации Reticulum
  -s SIZE, --size SIZE размер полезной нагрузки пакета зонда в байтах
  -n PROBES, --probes PROBES
                        количество отправляемых зондов
  -t seconds, --timeout seconds
                        таймаут до отказа
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  -w seconds, --wait seconds
                        время между каждым зондом
  --version             показать номер версии программы' и выйти
  -v, --verbose
```

3.2.6 Утилита rncp

Утилита rncp— это простой инструмент для передачи файлов. С ее помощью вы можете передавать файлы через Reticulum.

Примеры использования

Запустите rncp на принимающей системе, указав, каким идентификаторам разрешено отправлять файлы:

```
$ rncp --listen -a 1726dbad538775b5bf9b0ea25a4079c8 -a c50cc4e4f7838b6c31f60ab9032cbc62
```

Вы также можете указать разрешенные хеши идентификаторов (один на строку) в файле ~/.rncp /allowed_identities и просто запустить программу в режиме прослушивания:

```
$ rncp --listen
```

С другой системы скопируйте файл на принимающую систему:

```
$ rncp ~/path/to/file.tgz 73cbd378bb0286ed11a707c13447bb1e
```

Или получить файл с удалённой системы:

```
$ rncp --fetch ~/path/to/file.tgz 73cbd378bb0286ed11a707c13447bb1e
```

Все параметры командной строки

```
usage: rncp [-h] [--config path] [-v] [-q] [-S] [-l] [-F] [-f]
            [-j path] [-b seconds] [-a allowed_hash] [-n] [-p]
            [-w seconds] [--version] [file] [destination]

Reticulum File Transfer Utility

позиционные аргументы:
  file                  файл для передачи
  назначение            шестнадцатеричный хеш получателя

опции:
  -h, --help            показать это справочное сообщение и выйти
  --config path         путь к альтернативному каталогу конфигурации Reticulum
  -v, --verbose         увеличить детализацию
  -q, --quiet           уменьшить детализацию вывода
  -S, --silent          отключить вывод прогресса передачи
  -l, --listen          прослушивать входящие запросы на передачу
  -C, --no-compress     отключить автоматическое сжатие
  -F, --allow-fetch     разрешить аутентифицированным клиентам получать файлы
  -f, --fetch           получить файл от удаленного слушателя вместо отправки
  -j, --jail path       ограничить запросы на получение указанным путём
  -s, --save path       сохранять полученные файлы в указанном пути
  -O, --overwrite       Разрешить перезапись полученных файлов вместо добавления постфикса
  -b seconds            интервал анонсирования, 0 для анонсирования только при запуске
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  -a allowed_hash       разрешить эту идентификацию (или добавить в ~/.rncp/allowed_identities)
  -n, --no-auth         принимать запросы от кого угодно
  -p, --print-identity выводит информацию об идентификации и назначении и выходит
  -w seconds            таймаут отправителя до отказа
  -P, --phy-rates       отображает скорости передачи физического уровня
  --version             показать номер версии программы' и выйти
```

3.2.7 Утилита rnx

Утилита `rnx` — это базовая программа для удаленного выполнения команд. Она позволяет выполнять команды на удаленных системах через Reticulum и просматривать возвращенный вывод команд. Для полностью интерактивного решения удаленной оболочки обязательно также ознакомьтесь с программой rnsh.

Примеры использования

Запустите rnx на прослушивающей системе, указав, каким идентификаторам разрешено выполнять команды:

```
$ rnx --listen -a 941bed5e228775e5a8079fc38b1ccf3f -a 1b03013c25f1c2ca068a4f080b844a10
```

С другой системы выполните команду на удалённой:

```
$ rnx 7a55144adf826958a9529a3bcf08b149 "cat /proc/cpuinfo"
```

Или войдите в интерактивный режим псевдооболочки:

```
$ rnx 7a55144adf826958a9529a3bcf08b149 -x
```

Файл идентификации по умолчанию хранится в `~/.reticulum/identities/rnx` , но вы можете использовать другой, который будет создан, если он еще не существует.

```
$ rnx 7a55144adf826958a9529a3bcf08b149 -i /path/to/identity -x
```

Все параметры командной строки

```
usage: rnx [-h] [--config path] [-v] [-q] [-p] [-l] [-i identity] [-x] [-b] [-n] [-N]
           [-d] [-m] [-a allowed_hash] [-w seconds] [-W seconds] [--stdin STDIN]
           [--stdout STDOUT] [--stderr STDERR] [--version] [destination] [command]

Утилита удаленного выполнения Reticulum

позиционные аргументы:
  назначение            шестнадцатеричный хеш слушателя
  command               команда для выполнения

дополнительные аргументы:
  -h, --help            показать это справочное сообщение и выйти
  --config path         путь к альтернативному каталогу конфигурации Reticulum
  -v, --verbose         увеличить детализацию
  -q, --quiet           уменьшить детализацию вывода
  -p, --print-identity выводит информацию об идентификации и назначении и выходит
  -l, --listen          слушать входящие команды
  -i identity           путь к используемому идентификатору
  -x, --interactive     войти в интерактивный режим
  -b, --no-announce не объявлять при запуске программы
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  -a allowed_hash       принимать от этого идентификатора
  -n, --noauth          принимать файлы от кого угодно
  -N, --noid            не идентифицировать слушателя
  -d, --detailed        показать подробный вывод результатов
  -m                    зеркальный код выхода удаленной команды
  -w seconds            время ожидания подключения и запроса до отказа
  -W секунд             максимальное время загрузки результата
  --stdin STDIN         передать ввод в stdin
  --stdout STDOUT       максимальный размер возвращаемого stdout в байтах
  --stderr STDERR       максимальный размер возвращаемого stderr в байтах
  --version             показать номер версии программы' и выйти
```

3.2.8 Утилита rnodeconf

Утилита `rnodeconf` позволяет проверять и настраивать существующие *RNodes*, а также создавать и подготавливать новые *RNodes* с любых поддерживаемых аппаратных устройств.

Все параметры командной строки

```
usage: rnodeconf [-h] [-i] [-a] [-u] [-U] [--fw-version version]
                 [--fw-url url] [--nocheck] [-e] [-E] [-C]
                 [--baud-flash baud_flash] [-N] [-T] [-b] [-B] [-p] [-D i]
                 [--display-addr byte] [--freq Hz] [--bw Hz] [--txp dBm]
                 [--sf factor] [--cr rate] [--eeprom-backup] [--eeprom-dump]
                 [--eeprom-wipe] [-P] [--trust-key hexbytes] [--version] [-f]
                 [-r] [-k] [-S] [-H FIRMWARE_HASH] [--platform platform]
                 [--product product] [--model model] [--hwrev revision]
                 [порт]

Утилита для настройки и прошивки RNode. Эта программа позволяет изменять
различные настройки и режимы запуска RNode. Она также может устанавливать,
 прошивать и обновлять прошивку на поддерживаемых устройствах.

позиционные аргументы:
  порт                  последовательный порт, к которому подключен RNode

опции:
  -h, --help            показать это справочное сообщение и выйти
  -i, --info            Показать информацию об устройстве
  -a, --autoinstall     Автоматическая установка на различные поддерживаемые устройства
  -u, --update          Обновить прошивку до последней версии
  -U, --force-update    Обновить до указанной прошивки, даже если версия совпадает или старше
↪установленной версии
  --fw-version version Использовать определенную версию прошивки для обновления или автоустановки
  --fw-url url          Использовать альтернативный URL для загрузки прошивки
  --nocheck             Не проверять обновления прошивки онлайн
  -e, --extract         Извлечь прошивку из подключенного RNode для последующего использования
  -E, --use-extracted   Использовать извлеченную прошивку для автоустановки или обновления
  -C, --clear-cache     Очистить локально кэшированные файлы прошивки
  --baud-flash baud_flash
                        Установить конкретную скорость передачи данных при прошивке устройства. По умолчанию 921600
  -N, --normal          Переключить устройство в нормальный режим
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  -T, --tnc             Переключить устройство в режим TNC
  -b, --bluetooth-on    Включить Bluetooth на устройстве
  -B, --bluetooth-off   Выключить Bluetooth на устройстве
  -p, --bluetooth-pair Перевести устройство в режим сопряжения Bluetooth
  -D, --display i       Установить интенсивность дисплея (0-255)
  -t, --timeout s       Установить тайм-аут дисплея в секундах, 0 для отключения
  -R, --rotation rotation
                        Установить поворот дисплея, допустимые значения
  от 0 до 3 --display-addr byte Установить адрес дисплея как шестна-
  дцатеричный байт (00 - FF) --recondition-display
                        Начать восстановление дисплея
  --np i                Установить интенсивность NeoPixel (0-255)
  --freq Hz             Частота в Гц для режима TNC
  --bw Hz               Полоса пропускания в Гц для режима TNC
  --txp dBm             Мощность TX в дБм для режима TNC
  --sf factor           Коэффициент расширения для режима TNC (7 - 12)
  --cr rate             Скорость кодирования для режима TNC (5 - 8)
  -x, --ia-enable       Включить предотвращение помех
  -X, --ia-disable      Отключить предотвращение помех
  -c, --config          Распечатать конфигурацию устройства
  --eeprom-backup       Создать резервную копию EEPROM в файл
  --eeprom-dump         Вывести дамп EEPROM в консоль
  --eeprom-wipe         Разблокировать и очистить EEPROM
  -P, --public          Отобразить открытую часть ключа подписи
  --trust-key hexbytes Открытый ключ для доверия при проверке устройства
  --version             Вывести версию программы и выйти
  -f, --flash           Прошить микропрограмму и инициализировать EEPROM
  -r, --rom             Инициализировать EEPROM без прошивки микропрограммы
  -k, --key             Сгенерировать новый ключ подписи и выйти
  -S, --sign            Отобразить открытую часть ключа
  подписи -H, --firmware-hash FIRMWARE_HASH
                        Установить хеш прошивки
  --platform platform   Спецификация платформы для начальной загрузки устройства
  --product product     Спецификация продукта для начальной загрузки устройства
  --model model         Код модели для начальной загрузки устройства
  --hwrev revision      Ревизия оборудования для начальной загрузки устройства
```

Для получения дополнительной информации о том, как создать свои собственные RNodes, пожалуйста, прочтите раздел*Создание RNodes*этого руководства.

3.3 Удаленное управление

Возможно разрешить удаленное управление системами Reticulum, используя различные встроенные утилиты , такие как `rnstatus` и `rnpath` . Для этого вам потребуется установить директиву `enable_remote_management` в разделе `[reticulum]` конфигурационного файла. Вам также потребуется указать один или несколько хешей Reticulum Identity для аутентификации запросов от клиентских программ. Для этой цели вы можете использовать существующие файлы идентификации или сгенерировать новые с помощью утилиты rnid.

Ниже приведен сокращенный пример включения удаленного управления в файле конфигурации Reticulum:

```
[reticulum]
...
enable_remote_management = yes
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
remote_management_allowed = 9fb6d773498fb3feda407ed8ef2c3229,␣
↪2d882c5586e548d79b5af27bca1776dc
...
```

Для получения полного примера конфигурации вы можете запустить `rnsd --exampleconfig`.

3.4 Улучшение конфигурации системы

Если вы настраиваете систему для постоянного использования с Reticulum, существует несколько изменений в конфигурации системы, которые могут облегчить ее администрирование. Эти изменения будут подробно описаны здесь.

3.4.1 Фиксированные имена последовательных портов

В экземпляре Reticulum с несколькими интерфейсами, основанными на последовательных портах, может быть полезно использовать фиксированные имена устройств для последовательных портов вместо динамически назначаемых сокращений, таких как `/dev/ttyUSB0` . В большинстве дистрибутивов на основе Debian, включая Ubuntu и Raspberry Pi OS, эти узлы можно найти по пути `/dev/serial/by-id` .

Вы можете использовать такой путь к устройству непосредственно вместо пронумерованных сокращений. Вот пример пакетного радио TNC, настроенного таким образом:

```
[[Packet Radio KISS Interface]]
  type = KISSInterface
  interface_enabled = True
  outgoing = true
  port = /dev/serial/by-id/usb-FTDI_FT230X_Basic_UART_43891CKM-if00-port0
  speed = 115200
  databits = 8
  parity = none
  stopbits = 1
  preamble = 150
  txtail = 10
  persistence = 200
  slottime = 20
```

Использование этой методологии позволяет избежать потенциальной путаницы в именах, когда физические устройства могут быть подключены и отключены в разном порядке или когда назначение имени устройства меняется от одной загрузки к другой.

3.4.2 Reticulum как системная служба

Вместо того чтобы запускать Reticulum вручную, вы можете установить `rnsd` в качестве системной службы, чтобы она запускалась автоматически при загрузке.

Общесистемная служба

Если вы установили Reticulum с помощью `pip` , программа `rnsd` , скорее всего, будет находиться только в пользовательском локальном каталоге установки, что означает, что `systemd` не сможет ее выполнить. В этом случае вы можете просто создать символическую ссылку на программу `rnsd` в каталог, который находится в пути systemd:

```
sudo ln -s $(which rnsd) /usr/local/bin/
```

Затем вы можете создать файл службы `/etc/systemd/system/rnsd.service` со следующим содержимым:

```
[Unit]
Description=Демон сетевого стека Reticulum
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
After=multi-user.target

[Service]
# Если вы запускаете Reticulum на устройствах Wi-
Fi # или других устройствах, которым требуется
дополнительное # время для инициализации,
вы можете # добавить небольшую задержку перед
запуском Reticulum # systemd:
# ExecStartPre=/bin/sleep 10
Type=simple
Restart=always
RestartSec=3
User=USERNAMEHERE
ExecStart=rnsd --service

[Install]
WantedBy=multi-user.target
```

Обязательно замените USERNAMEHERE на имя пользователя, от которого вы хотите запускать rnsd.

Чтобы запустить rnsdвручную, выполните:

```
sudo systemctl start rnsd
```

Если вы хотите автоматически запускать rnsdпри загрузке, выполните:

```
sudo systemctl enable rnsd
```

Сервис пользовательского пространства

В качестве альтернативы можно использовать пользовательский сервис systemd вместо общесистемного . Таким образом, вся настройка может быть выполнена как обычный пользователь. Создайте файл пользовательского сервиса systemd ~/.config/systemd/user/rnsd.service со следующим содержимым:

```
[Unit]
Description=Reticulum Network Stack Daemon
After=default.target

[Service]
# Если вы запускаете Reticulum на устройствах Wi-
Fi # или других устройствах, которым требуется
дополнительное # время для инициализации,
вы можете # добавить небольшую задержку перед
запуском Reticulum # systemd:
# ExecStartPre=/bin/sleep 10
Type=simple
Restart=always
RestartSec=3
ExecStart=RNS_BIN_DIR/rnsd --service

[Install]
WantedBy=default.target
```

ЗаменитеRNS_BIN_DIRна путь к вашему бинарному каталогу Reticulum (например, /home/USERNAMEHERE/rns/bin).

Запуск пользовательского сервиса:

```
systemctl --user daemon-reload
systemctl --user start rnsd.service
```

Если вы хотите автоматически запускать `rnsd`без необходимости входа в систему под именем USERNAMEHERE, сделайте следующее:

```
sudo loginctl enable-linger USERNAMEHERE
systemctl --user enable rnsd.service
```

ГЛАВА

ЧЕТЫРЕ

ПОНИМАНИЕ RETICULUM

Эта глава кратко опишет общую цель и принципы работы Reticulum. Она должна дать вам общее представление о функционировании стека, а также понимание того, как разрабатывать сетевые приложения с использованием Reticulum.

Эта глава не является исчерпывающим источником информации о Reticulum, по крайней мере, пока. В настоящее время единственным полным репозиторием и окончательным авторитетом в отношении фактического функционирования Reticulum является эталонная реализация на Python и ссылка на API. Несмотря на это, данная глава является важным ресурсом для понимания принципов работы Reticulum на высоком уровне, а также общих принципов Reticulum и способов их применения при создании собственных сетей или программного обеспечения.

Прочитав этот документ, вы будете хорошо подготовлены к пониманию того, как функционирует сеть Reticulum, что она может достичь и как вы можете использовать ее самостоятельно. Если вы хотите помочь в разработке, это также хорошее место для начала, поскольку оно даст довольно четкое представление о настроениях и философии Reticulum, о том, какие проблемы он стремится решить, и как он подходит к этим решениям.

4.1 Мотивация

Основной мотивацией для разработки и внедрения Reticulum является текущий недостаток надежных, функциональных и безопасных режимов цифровой связи, требующих минимальной инфраструктуры. Я считаю, что крайне желательно создать надежный и эффективный способ построения цифровых сетей дальней связи, которые могут безопасно обеспечивать обмен информацией между людьми и машинами, без централизованной точки власти, контроля, цензуры или барьера для входа.

Почти все используемые сегодня сетевые системы имеют общее ограничение: для их функционирования требуется большое количество координации, централизованного доверия и власти. Чтобы присоединиться к таким сетям, вам необходимо одобрение контролирующих их лиц. Эта потребность в координации и доверии неизбежно приводит к среде централизованного контроля, где операторам инфраструктуры или правительствам очень легко контролировать или изменять трафик, а также подвергать цензуре или преследовать нежелательных участников. Это также делает абсолютно невозможным свободное развертывание и использование сетей по собственному желанию, как это делается с другими распространёнными инструментами, расширяющими индивидуальные возможности и свободу.

Reticulum стремится требовать как можно меньше координации и доверия. Его цель — сделать безопасную, анонимную и не требующую разрешений сеть и обмен информацией инструментом, который любой может просто взять и использовать.

Поскольку Reticulum полностью не зависит от среды передачи, его можно использовать для построения сетей на любом носителе, который лучше всего подходит для ситуации или который у вас есть в наличии. В некоторых случаях это могут быть пакетные радиолинии на ОВЧ-частотах, в других — сеть 2,4 ГГц с использованием готовых радиостанций, или же это могут быть обычные платы для разработки LoRa.

На момент выпуска этого документа самая быстрая и простая настройка для разработки и тестирования — это использование радиомодулей LoRa с прошивкой с открытым исходным кодом (см. раздел *Reference Setup*), подключённых к любому типу компьютера или мобильного устройства, на котором может работать Reticulum.

Конечная цель Reticulum — позволить каждому быть своим собственным сетевым оператором, а также сделать дешевым и легким охват обширных территорий множеством независимых, взаимосвязанных и автономных сетей. Reticulum — это не одна

сеть , это инструмент для создания *тысяч сетей* . Сети без аварийных выключателей, слежки, цензуры и контроля. Сети, которые могут свободно взаимодействовать, ассоциироваться и разъединяться друг с другом и не требуют централизованного надзора.
Сети для людей.*Сети для народа*.

4.2 Цели

Чтобы быть максимально широко используемым и эффективным в развертывании, при разработке Reticulum руководствовались следующими целями:

- **Полностью пригодный для использования в качестве стека программного обеспечения с открытым исходным кодом**
  Reticulum должен быть реализован и способен работать только с использованием программного обеспечения с открытым исходным кодом. Это критически важно для обеспечения доступности, безопасности и прозрачности системы.
- **Агностицизм аппаратного уровня**
  Reticulum должен быть полностью аппаратно-независимым и пригодным для использования поверх широкого спектра физических сетевых уровней, таких как радиомодемы, последовательные линии, модемы, портативные трансиверы, проводной Ethernet, WiFi или что-либо еще, что может передавать цифровой поток данных. Аппаратное обеспечение, разработанное для выделенного использования Reticulum, должно быть максимально дешевым и использовать готовые компоненты, чтобы его мог легко модифицировать и воспроизвести любой заинтересованный в этом.
- **Очень низкие требования к пропускной способности**
  Reticulum должен быть способен надежно функционировать на каналах с пропускной способностью всего *5 бит в секунду* .
- **Шифрование по умолчанию**
  Reticulum должен использовать сильное шифрование по умолчанию для всей связи.
- **Анонимность инициатора**
  Должна быть возможность общаться по сети Reticulum, не раскрывая никакой идентифицирующей информации о себе.
- **Использование без лицензии**
  Reticulum должен функционировать поверх физических средств связи, которые не требуют какой-либо формы лицензии для использования. Reticulum должен быть разработан таким образом, чтобы его можно было использовать в радиочастотных диапазонах ISM, и он мог обеспечивать функциональные междугородние соединения в таких условиях, например, путем подключения модема к PMR или CB-радио, или с помощью модулей LoRa или WiFi.
- **Поставляемое программное обеспечение**
  В дополнение к основному сетевому стеку и API, которые позволяют разработчику создавать приложения с Reticulum, базовый набор инструментов связи на основе Reticulum должен быть реализован и выпущен вместе с самим Reticulum. Они должны служить как функциональным, базовым набором для связи, так и примером и обучающим ресурсом для тех, кто желает создавать приложения с Reticulum.
- **Простота использования**
  Эталонная реализация Reticulum написана на Python, чтобы ее было легко использовать и понимать. Программист с базовым опытом должен быть в состоянии использовать Reticulum для написания сетевых приложений.
- **Низкая стоимость**
  Развертывание системы связи на основе Reticulum должно быть как можно более дешевым. Это должно быть достигнуто за счет использования дешёвого готового оборудования, которое потенциальные пользователи, возможно, уже имеют. Стоимость настройки функционирующего узла должна быть менее $100, даже если все части необходимо приобрести.

4.3 Введение и базовая функциональность

Reticulum — это сетевой стек, подходящий для каналов с высокой задержкой и низкой пропускной способностью. Reticulum по своей сути является системой, *ориентированной на сообщения* . Он подходит как для локальных двухточечных, так и для многоточечных сценариев, где все узлы находятся в пределах

диапазона друг от друга, а также для сценариев, где пакеты должны быть транспортированы через несколько переходов в сложной сети для достижения получателя.

Reticulum отказывается от идеи адресов и портов, известных из IP, TCP и UDP. Вместо этого Reticulum использует единственную концепцию *назначений* . Любое приложение, использующее Reticulum в качестве своего сетевого стека, должно будет создать одно или несколько назначений для получения данных и знать назначения, которым оно должно отправлять данные.

Все назначения в Reticulum _представлены_ в виде 16-байтового хеша. Этот хеш получен путем усечения полного хеша SHA-256 идентифицирующих характеристик назначения. Для пользователей адреса назначения будут отображаться как 16 шестнадцатеричных байтов, как в этом примере: `<13425ec15b621c1d928589718000d814>` .

Размер усечения в 16 байт (128 бит) для адресов был выбран как разумный компромисс между адресным пространством и накладными расходами на пакеты. Адресное пространство, обеспечиваемое этим размером, может поддерживать миллиарды одновременно активных устройств в одной сети, при этом сохраняя низкие накладные расходы на пакеты, что крайне важно для сетей с низкой пропускной способностью В крайне маловероятном случае, если это адресное пространство приблизится к перегрузке, однострочное изменение кода может обновить адресное пространство Reticulum вплоть до 256 бит, гарантируя, что оно потенциально сможет поддерживать сети галактического масштаба. Это, очевидно, полное и нелепое избыточное выделение, и поэтому текущих 128 бит должно быть достаточно даже в далеком будущем.

По умолчанию Reticulum шифрует все данные, используя криптографию на эллиптических кривых и AES. Любой пакет, отправленный получателю, шифруется с помощью ключа, полученного для каждого пакета. Reticulum также может установить зашифрованный канал с получателем, называемый *Link* . Как данные, отправляемые через Links, так и отдельные пакеты обеспечивают *анонимность инициатора* . Links дополнительно обеспечивает прямую секретность по умолчанию, используя обмен ключами Диффи-Хеллмана на эллиптических кривых на Curve25519 для получения эфемерных ключей для каждого соединения. Асимметричная, бессвязная пакетная связь также может обеспечивать прямую секретность с автоматическим обновлением ключей путем включения обновлений для каждого пункта назначения. Многохоповые транспортные уровни, а также уровни координации, верификации и надёжности полностью автономны и также основаны на криптографии с эллиптическими кривыми.

Reticulum также предлагает симметричное шифрование ключей для групповых коммуникаций, а также незашифрованные пакеты для целей локального широковещания.

Reticulum может подключаться к различным интерфейсам, таким как радиомодемы, радиостанции передачи данных и последовательные порты, и предлагает возможность легко туннелировать трафик Reticulum через IP-каналы, такие как Интернет или частные IP-сети.

4.3.1 Назначения

Для получения и отправки данных с помощью стека Reticulum приложению необходимо создать одно или несколько назначений . Reticulum использует три различных базовых типа адресатов и один специальный:

- **Single**

  Тип адресата *single*является наиболее распространенным в Reticulum и должен использоваться в большинстве случаев. Он всегда идентифицируется уникальным открытым ключом. Любые данные, отправленные этому адресату, будут зашифрованы с использованием эфемерных ключей, полученных в результате обмена ключами ECDH, и будут доступны для чтения только создателю адресата, который владеет соответствующим закрытым ключом.

- **Plain**

  Тип адресата *plain* не зашифрован и подходит для трафика, который должен быть передан большому количеству пользователей или быть доступным для чтения всем. Трафик к адресату *plain* не зашифрован. В целом, адресаты *plain* могут использоваться для широковещательной информации, предназначенной для публичного доступа. Адресаты Plain доступны только напрямую, и пакеты, адресованные адресатам plain, никогда не передаются через несколько переходов в сети. Чтобы быть переносимой через несколько переходов в Reticulum, информация должна быть зашифрована, поскольку Reticulum использует шифрование для каждого пакета для проверки маршрутов и поддержания их активности.

- **Группа**

  Специальный тип назначения группы, который определяет симметрично зашифрованное виртуальное назначение . Данные, отправленные в это назначение, будут зашифрованы симметричным ключом и будут читаемы любым, кто обладает ключом, но, как и в случае с обычным типом назначения, пакеты для этого типа назначения в настоящее время не передаются через несколько хопов, хотя планируемое обновление Reticulum позволит глобально достижимые групповые назначения.

- **Связь**

  Связь — это специальный тип назначения, который служит абстрактным каналом к одному назначению, напрямую подключенному или через несколько хопов. Связь также предлагает надежность и более эффективное шифрование, прямую секретность, анонимность инициатора, и поэтому может быть полезна даже когда узел напрямую достижим. Он также предлагает более мощный API и позволяет легко выполнять запросы и ответы, передавать большие объемы данных и многое другое.

Именование назначений

Назначения создаются и именуются в легко понятной точечной нотации аспектов и представляются в сети как хеш этого значения. Хеш представляет собой SHA-256, усечённый до 128 бит. Аспект верхнего уровня всегда должен быть уникальным идентификатором для приложения, использующего назначение . Следующие уровни аспектов могут быть определены создателем приложения любым способом.

Аспекты могут быть сколь угодно длинными и многочисленными, и результирующее длинное имя назначения не повлияет на эффективность, поскольку имена в сети всегда представлены в виде усечённых хешей SHA-256.

Например, назначение для приложения мониторинга окружающей среды может состоять из имени приложения, типа устройства и типа измерения, например, так:

```
имя приложения : environmentlogger
аспекты : remotesensor, temperature

полное имя : environmentlogger.remotesensor.temperature
hash       : 4faf1b2e0a077e6a9d92fa051f256038
```

Для единичного назначения Reticulum автоматически добавит связанный открытый ключ в качестве аспекта назначения перед хешированием. Это делается для того, чтобы было достигнуто только правильное назначение, поскольку любой может прослушивать любое имя назначения. Добавление открытого ключа гарантирует, что данный пакет будет направлен только тому адресату, который владеет соответствующим закрытым ключом для его расшифровки.

Обратите внимание!Здесь есть очень важная концепция, которую необходимо понять:

- Любой может использовать имя адресатаenvironmentlogger.remotesensor.temperature
- Каждый адресат, поступающий таким образом, всё равно будет иметь уникальный хеш адресата и, таким образом, будет уникально адресуемым, поскольку их открытые ключи будут различаться.

При фактическом использованииединогоименования адресатов не рекомендуется использовать какие-либо уникальные идентифицирующие признаки в именовании аспектов. Имена аспектов должны быть общими терминами, описывающими тип представленного адресата. Уникально идентифицирующий аспект всегда достигается путём добавления открытого ключа, который расширяет адресата до уникально идентифицируемого. Reticulum делает это автоматически.

Любой адресат в сети Reticulum может быть адресован и достигнут, просто зная его хеш адресата (и открытый ключ; но если открытый ключ неизвестен, его можно запросить у сети, просто зная хеш адресата). Использование имён приложений и аспектов упрощает структурирование программ Reticulum и позволяет фильтровать информацию и данные, которые получает ваша программа.

Подытоживая, различные типы адресов назначения следует использовать в следующих ситуациях:

- **Single**

  Когда требуется приватная связь между двумя конечными точками. Поддерживает несколько переходов.

- **Группа**

  Когда требуется приватная связь между двумя или более конечными точками. Косвенно поддерживает несколько переходов, но сначала должен быть установлен через один адрес назначения.

- **Plain**

  Когда желательна передача открытого текста, например, при широковещательной передаче информации или для целей локального обнаружения.

Чтобы связаться с одним адресом назначения, вам нужно знать его открытый ключ. Любой метод получения открытого ключа действителен, но Reticulum включает простой механизм для информирования других узлов об открытом ключе вашего адреса назначения, называемый анонсом . Также возможно запросить неизвестный открытый ключ из сети, поскольку все экземпляры транспортного протокола служат распределенным реестром открытых ключей.

Обратите внимание, что информация об открытом ключе может быть передана и проверена другими способами, помимо использования встроенной функции анонса , и поэтому не требуется использовать функции анонса и *запроса пути* для получения открытых ключей. Однако это, безусловно, самый простой способ, и его определенно следует использовать, если нет очень веской причины поступать иначе.

4.3.2 Объявления открытых ключей

Объявление отправит специальный пакет по любым соответствующим интерфейсам, содержащий всю необходимую информацию о хеше назначения и открытом ключе, а также может содержать некоторые дополнительные, специфичные для приложения данные. Весь пакет подписывается отправителем для обеспечения подлинности. Использовать функцию объявления не требуется, но во многих случаях это будет самый простой способ обмениваться открытыми ключами в сети. Механизм объявления также служит для установления сквозного соединения с объявленным назначением по мере распространения объявления по сети.

Например, объявление в простом приложении-мессенджере может содержать следующую информацию:

- Хеш назначения объявляющего
- Открытый ключ объявляющего
- Данные, специфичные для приложения, в данном случае никнейм пользователя и статус доступности
- Случайный набор данных, делающий каждое новое объявление уникальным
- Подпись Ed25519 вышеуказанной информации, подтверждающая подлинность

Благодаря этой информации любой узел Reticulum, который ее получит, сможет реконструировать исходящий пункт назначения для безопасной связи с ним. Вы могли заметить, что для полного восстановления данных об объявленном пункте назначения не хватает одной части информации, а именно — имен аспектов пункта назначения. Они намеренно опущены для экономии пропускной способности, поскольку почти во всех случаях будут подразумеваться. Принимающее приложение уже будет их знать. Если имя пункта назначения не полностью подразумевается, информация может быть включена в часть данных, специфичных для приложения, что позволит получателю вывести именование.

Важно отметить, что объявления будут пересылаться по сети в соответствии с определённым шаблоном. Это будет подробно описано в разделе *Механизм объявления подробно* .

В Reticulum пункты назначения могут свободно перемещаться по сети. Это сильно отличается от таких протоколов, как IP, где адрес всегда должен оставаться в том сетевом сегменте, в котором он был назначен. В Reticulum этого ограничения не существует, и любой пункт назначения *полностью переносим* по всей топологии сети и *может быть даже перемещён в другие сети* Reticulum , отличные от той, в которой он был создан, и при этом оставаться доступным. Чтобы обновить свою достижимость, адресату достаточно отправить объявление в любой сети, частью которой он является. Через короткое время он станет глобально достижимым в сети.

Понимание того, какотдельныеадресаты всегда привязаны к паре приватного/публичного ключей, подводит нас к следующей теме.

4.3.3 Идентификаторы

В Reticulum идентификатор не обязательно представляет личную идентификацию, а является абстракцией, которая может представлять любой вид *проверяемой сущности* . Это вполне может быть человек, но также это может быть интерфейс управления машиной, программа, робот, компьютер, датчик или что-то совершенно другое. В целом, любой вид агента, который может действовать или над которым можно действовать, а также хранить или манипулировать информацией, может быть представлен как идентификатор. An идентификатор can be used to create any number of destinations.

Одно место назначения всегда будет иметь идентификатор , привязанный к нему, но не простые или групповые места назначения. Пункты назначения и идентификаторы имеют многостороннюю связь. Вы можете создать пункт назначения, и если он не связан с идентификатором при создании, то автоматически будет создан новый для использования. Это может быть желательно в некоторых ситуациях, но чаще всего вы, вероятно, захотите сначала создать идентификатор, а затем использовать его для создания новых пунктов назначения.

Например, мы могли бы использовать идентификатор для представления пользователя приложения для обмена сообщениями. Затем этот идентификатор может создавать пункты назначения, чтобы обеспечить связь с пользователем. Во всех случаях крайне важно надёжно и конфиденциально хранить закрытые ключи, связанные с любым идентификатором Reticulum, поскольку получение доступа к ключам идентификатора равносильно получению доступа и контролю достижимости любых пунктов назначения, созданных этим идентификатором.

4.3.4 Дальнейшие шаги

Приведённые выше функции и принципы формируют ядро Reticulum и будут достаточными для создания функциональных сетевых приложений в локальных кластерах, например, по радиоканалам, где все заинтересованные узлы могут напрямую слышать друг друга. Но чтобы быть по-настоящему полезным, нам нужен способ направлять трафик через несколько переходов в сети.

В следующих разделах будут представлены две концепции, которые это позволяют:путиилинки.

4.4 Транспорт Reticulum

Методы маршрутизации, используемые в традиционных сетях, принципиально несовместимы с типами физических сред и условиями, для работы с которыми был разработан Reticulum. Эти механизмы в основном предполагают доверие на физическом уровне и часто требуют значительно большей пропускной способности, чем та, которую Reticulum может считать доступной. Поскольку Reticulum разработан для работы в открытом радиочастотном спектре, такое доверие невозможно предположить, а пропускная способность часто очень ограничена.

Для преодоления таких проблем транспортная система Reticulum использует асимметричную криптографию на эллиптических кривых для реализации концепции путей , которые позволяют определять, как доставлять информацию ближе к определенному месту назначения. Важно отметить, что ни один отдельный узел в сети Reticulum не знает полного пути к месту назначения. Каждый транспортный узел, участвующий в сети Reticulum, будет знать только наиболее прямой способ доставить пакет на один шаг ближе к его месту назначения.

4.4.1 Типы узлов

В настоящее время Reticulum различает два типа сетевых узлов. Все узлы в сети Reticulum являются э*кземплярами* Reticulum , а некоторые также являются *транспортными узлами* . Если система, работающая под управлением Reticulum, закреплена в одном месте и предназначена для постоянной доступности, она является хорошим кандидатом на роль *транспортного узла* .

Любой экземпляр Reticulum может стать транспортным узлом, включив его в конфигурации. Это различие определяется пользователем, настраивающим узел, и используется для определения того, какие узлы в сети будут перенаправлять трафик, а какие узлы зависят от других узлов для более широкой связности.

Если узел является экземпляром , ему следует присвоить директиву конфигурации `enable_transport = No` , что является настройкой по умолчанию.

Если это*транспортный узел*, ему следует присвоить директиву конфигурации `enable_transport = Yes`.

4.4.2 Механизм анонсирования в деталях

Когда объявление о назначении передается экземпляром Reticulum, оно будет пересылаться любым получающим его транспортным узлом, но согласно следующим конкретным правилам:

- Если такое объявление уже было получено ранее, оно игнорируется.
- В противном случае в таблицу записываются сведения о транспортном узле, от которого было получено объявление, и общее количество его ретрансляций до текущего узла.
- Если объявление было ретранслировано $m+1$ раз, оно больше не будет пересылаться. По умолчанию m установлено значение 128.
- После случайной задержки объявление будет ретранслировано на всех интерфейсах, имеющих доступную пропускную способность для обработки объявлений. По умолчанию максимальное выделение пропускной способности для обработки объявлений составляет 2%, но может быть настроено для каждого интерфейса отдельно.

- Если какой-либо интерфейс не имеет достаточной пропускной способности для ретрансляции объявления, ему будет присвоен приоритет, обратно пропорциональный его счетчику переходов, и оно будет помещено в очередь, управляемую этим интерфейсом.
- Когда интерфейс имеет доступную пропускную способность для обработки объявлений, он будет приоритизировать объявления для ближайших по количеству переходов адресатов, тем самым приоритизируя доступность и связность локальных узлов, даже в медленных сетях, подключающихся к более широким и быстрым сетям.
- После повторной передачи объявления, и если не будет зафиксировано, что другие узлы повторно передают объявление с большим количеством переходов, чем при его отправке с данного узла, попытка передачи будет повторена r раз. По умолчанию r установлено значение 1.
- Если поступает более новое объявление от того же адресата, в то время как идентичное уже ожидает передачи, самое новое объявление отбрасывается. Если самое новое объявление содержит другие специфические для приложения данные, оно заменит старое объявление.

Как только объявление достигает узла в сети, любой другой узел, находящийся в прямом контакте с этим узлом, сможет достичь адресата, откуда исходило объявление, просто отправив пакет, адресованный этому адресату. Любой узел, знающий об объявлении, сможет направить пакет к месту назначения, найдя следующий узел с наименьшим количеством переходов до места назначения.

Согласно этим правилам, объявление будет распространяться по сети предсказуемым образом, делая объявленное назначение доступным за короткое время. Быстрые сети, способные обрабатывать множество объявлений, могут достигать полной сходимости очень быстро, даже при постоянном добавлении новых назначений. Более медленным сегментам таких сетей может потребоваться немного больше времени для получения полной информации о широких и быстрых сетях, к которым они подключены, но они всё равно смогут это сделать со временем, приоритизируя полную и быстро сходящуюся сквозную связь для своих локальных, более медленных сегментов.

В целом, даже чрезвычайно сложные сети, использующие максимум 128 хопов, будут сходиться к полной сквозной связи примерно за одну минуту, при условии наличия достаточной пропускной способности для обработки необходимого количества объявлений.

4.4.3 Достижение места назначения

В сетях с изменяющейся топологией и ненадежным соединением узлам необходим способ установления *проверенного соединения* друг с другом. Поскольку предполагается, что сеть является ненадежной, Reticulum должен предоставить способ гарантировать, что пир, с которым вы общаетесь, на самом деле тот, кого вы ожидаете. Reticulum предлагает два способа сделать это.

Для обмена небольшими объемами информации Reticulum предлагает *Packet* API, который работает именно так, как ожидается, — на уровне отдельных пакетов. При отправке пакета используется следующий процесс:

- Пакет всегда создается с связанным адресом назначения и определенными данными полезной нагрузки. При отправке пакета в один тип назначения Reticulum автоматически создает эфемерный ключ шифрования, выполняет обмен ключами ECDH с открытым ключом назначения (или ключом храповика, если он доступен) и шифрует информацию.
- Важно отметить, что этот обмен ключами не требует сетевого трафика. Отправитель уже знает открытый ключ получателя из ранее полученного объявления и, таким образом, может выполнить обмен ключами ECDH локально, прежде чем отправить пакет.
- Открытая часть недавно сгенерированной эфемерной пары ключей включается в зашифрованный токен и отправляется вместе с зашифрованными данными полезной нагрузки в пакете.
- Когда пункт назначения получает пакет, он сам может выполнить обмен ключами ECDH и расшифровать пакет.
- Новый эфемерный ключ используется для каждого пакета, отправляемого таким образом.
- После того как пакет был получен и расшифрован адресатом, этот адресат может выбрать подтверждение его получения. Это делается путем вычисления хеша SHA-256 полученного пакета и подписи этого хеша своим ключом подписи Ed25519. Транспортные узлы в сети могут затем направить это доказательство обратно к источнику пакета, где подпись может быть проверена по известному открытому ключу подписи адресата.
- В случае, если пакет адресован групповому типу назначения, он будет зашифрован предварительно совместно используемым ключом AES-256, связанным с этим назначением. В случае, если пакет адресован открытому типу назначения,

данные полезной нагрузки не будут зашифрованы. Ни один из этих двух типов назначения не может обеспечить прямую секретность. В общем, рекомендуется всегда использовать одиночный тип назначения, если только не требуется строго использовать один из других.

Для обмена большими объемами данных или когда требуются более длительные сеансы двунаправленной связи, Reticulum предлагает API *Link* . Для установления связи используется следующий процесс:

- Во-первых, узел, желающий установить связь, отправит специальный пакет, который пройдет по сети и найдет желаемое назначение. По пути транспортные узлы, которые пересылают пакет, примут к сведению этот *запрос связи* .
- Во-вторых, если назначение принимает *запрос связи* , оно отправит обратно пакет, который доказывает подлинность его личности (и получение запроса связи) инициирующему узлу. Все узлы, которые изначально пересылали пакет, также смогут проверить это доказательство и, таким образом, принять действительность связи по всей сети.
- Когда действительность связи была принята пересылающими узлами, эти узлы запомнят связь , и впоследствии ее можно будет использовать, ссылаясь на хеш, представляющий ее.
- В рамках запроса *link* происходит обмен ключами Диффи-Хеллмана на эллиптических кривых, который устанавливает эффективно зашифрованный туннель между двумя узлами. Таким образом, этот режим связи предпочтителен даже в ситуациях, когда узлы могут напрямую обмениваться данными, когда объём обмениваемых данных исчисляется десятками пакетов или когда желательно использование более продвинутых функций API.
- Когда установлена связь , она автоматически обеспечивает функцию получения сообщений посредством того же механизма подтверждения, обсуждавшегося ранее, так что отправляющий узел может получить проверенное подтверждение того, что информация достигла предполагаемого получателя.
- После установки связи инициатор может оставаться анонимным или выбрать аутентификацию по отношению к адресату, используя идентификатор Reticulum. Эта аутентификация происходит внутри зашифрованной связи и раскрывается только проверенному адресату, а не посредникам.

Через мгновение мы обсудим детали того, как реализована эта методология, но сначала давайте вспомним, каким целям она служит. Сначала мы убеждаемся, что узел, отвечающий на наш запрос, действительно является тем, с кем мы хотим общаться, а не злоумышленником, притворяющимся таковым. Одновременно мы устанавливаем эффективный зашифрованный канал. Настройка этого относительно дёшева с точки зрения пропускной способности, поэтому её можно использовать только для короткого обмена , а затем воссоздавать по мере необходимости, что также будет ротировать ключи шифрования. Соединение также может поддерживаться в течение более длительных периодов времени, если это больше подходит для приложения. Процедура также вставляет идентификатор соединения — хеш, рассчитанный из пакета запроса соединения, — в память узлов пересылки, что означает, что обменивающиеся узлы могут после этого достигать друг друга, просто ссылаясь на этот идентификатор соединения .

Общая стоимость пропускной способности для установки соединения составляет 3 пакета общим объемом 297 байт (более подробная информация в разделе Двоичный формат пакетов). Объём пропускной способности, используемый для поддержания открытого соединения, практически незначителен и составляет 0,45 бит в секунду. Даже на медленном радиоканале со скоростью 1200 бит в секунду 100 одновременных соединений всё равно оставят 96% пропускной способности канала для фактических данных.

Установление соединения в деталях

После изучения основ механизма анонсирования, поиска пути через сеть и общего обзора процедуры установления соединения, в этом разделе будет более подробно рассмотрен процесс установления соединения в Reticulum.

Соединение в терминологии Reticulum следует рассматривать не как прямое соединение между узлами на физическом уровне, а как абстрактный канал, который может быть открыт в течение любого времени и может охватывать произвольное количество переходов, где информация будет обмениваться между двумя узлами.

- Когда узел в сети хочет установить проверенное соединение с другим узлом, он случайным образом генерирует новую пару закрытого и открытого ключей X25519. Затем он создаёт *пакет запроса на соединение* и транслирует его.

Следует отметить, что упомянутая выше пара открытого/закрытого ключей X25519 состоит из двух отдельных пар ключей: пары ключей шифрования, используемой для вывода общего симметричного ключа, и пары ключей подписи, используемой для подписи и

проверка сообщений на канале. Они отправляются вместе по сети и для простоты данного объяснения могут рассматриваться как единый открытый ключ.

- Запрос на установку*канала*адресуется хешу желаемого получателя и содержит следующие данные: Недавно сгенерированный открытый ключ X25519 LKi.
- Широковещательный пакет будет направлен через сеть в соответствии с ранее установленными правилами.
- Любой узел, который пересылает запрос на установку канала, будет хранить идентификатор канала в своей таблице каналов , наряду с количеством переходов, которые пакет совершил при получении. Идентификатор канала является хешем всего пакета запроса на установку канала. Если пакет запроса на установку канала не будет подтвержден адресованным получателем в течение некоторого установленного времени, запись будет снова удалена из таблицы каналов.
- Когда получатель принимает пакет запроса на установку канала, он принимает решение о принятии запроса . В случае принятия, получатель также генерирует новую пару закрытого/открытого ключа X25519 и выполняет обмен ключами Диффи-Хеллмана, выводя новый симметричный ключ, который будет использоваться для шифрования канала после его установления.
- Пакет подтверждения связи теперь формируется и передается по сети. Этот пакет адресован идентификатору связи . Он содержит следующие данные: вновь сгенерированный открытый ключ X25519 LKr и подпись Ed2551 9 идентификатора связи и LKr , сделанную исходным ключом подписи адресованного назначения.
- Проверив этот пакет подтверждения связи , все узлы, которые изначально транспортировали пакет *запроса связи* к месту назначения от отправителя, теперь могут убедиться, что предполагаемое место назначения получило запрос и приняло его, и что выбранный ими путь для пересылки запроса был действительным. Успешно проведя эту проверку, транспортирующие узлы помечают связь как активную. Абстрактный двунаправленный канал связи теперь установлен вдоль пути в сети. Пакеты теперь могут обмениваться двунаправленно с любого конца связи, просто адресовав пакеты идентификатору связи .
- Когда источник получает подтверждение , он будет недвусмысленно знать, что проверенный путь к месту назначения установлен. Теперь он также может использовать открытый ключ X25519, содержащийся в подтверждении ссылки , для выполнения собственного обмена ключами Диффи-Хеллмана и получения симметричного ключа, используемого для шифрования канала. Теперь информацией можно обмениваться надёжно и безопасно.

Важно отметить, что эта методология гарантирует, что источник запроса не должен раскрывать никакой идентифицирующей информации о себе. Инициатор ссылки остаётся полностью анонимным.

При использовании ссылок Reticulum автоматически проверяет все данные, отправленные по ссылке, а также может автоматизировать повторные передачи, если используются ресурсы .

4.4.4 Ресурсы

Для обмена небольшими объёмами данных по сети Reticulum достаточно интерфейса *Packet* , но для обмена данными, требующими множества пакетов, необходим эффективный способ координации передачи.

В этом заключается назначение ресурса Reticulum. Ресурс может автоматически обрабатывать надёжную передачу произвольного объёма данных по установленному соединению . Ресурсы могут автоматически сжимать данные, обрабатывать их разбиение на отдельные пакеты, упорядочивать передачу, проверять целостность и повторно собирать данные на другом конце.

Ресурсы программно очень просты в использовании и требуют всего нескольких строк кода для надёжной передачи любого объёма данных. Их можно использовать для передачи данных, хранящихся в памяти, или для потоковой передачи данных непосредственно из файлов.

4.5 Эталонная установка

В этом разделе будет подробно описана рекомендуемая *эталонная установка* для Reticulum. Важно отметить, что Reticulum разработан для использования практически на любом вычислительном устройстве и через практически любую среду, позволяющую отправлять и получать данные, которая удовлетворяет очень низким минимальным требованиям

Канал связи должен поддерживать по крайней мере полудуплексный режим работы, обеспечивать среднюю пропускную способность 5 бит в секунду или более и поддерживать MTU физического уровня 500 байт. Стек Reticulum должен быть способен работать практически на любом оборудовании, которое может предоставить среду выполнения Python 3.x.

Тем не менее, эта эталонная конфигурация была разработана для обеспечения общей платформы для всех, кто хочет помочь в разработке Reticulum, а также для всех, кто хочет узнать рекомендуемую конфигурацию для начала экспериментов. Эталонная система состоит из трех частей:

- **Интерфейсное устройство**
  Которое обеспечивает доступ к физической среде, по которой происходит связь, например, радиостанция со встроенным модемом. Конфигурация с отдельным модемом, подключенным к радиостанции, также будет являться интерфейсным устройством.
- **Хост-устройство**
  Некоторое вычислительное устройство, которое может запускать необходимое программное обеспечение, взаимодействовать с интерфейсным устройством и обеспечивать взаимодействие с пользователем.
- **Программный стек**
  Программное обеспечение, реализующее протокол Reticulum, и приложения, использующие его.

Эталонную конфигурацию можно рассматривать как относительно стабильную платформу для разработки, а также для начала построения сетей или приложений. Хотя детали реализации могут меняться на текущем этапе разработки, целью является поддержание аппаратной совместимости как можно дольше, и текущая эталонная конфигурация была определена как функциональная платформа на многие годы вперёд. Текущая эталонная конфигурация системы выглядит следующим образом:

- **Интерфейсное устройство**
  Радиомодуль передачи данных, состоящий из радиомодуля LoRa и микроконтроллера с прошивкой с открытым исходным кодом, который может подключаться к хост-устройствам через USB. Он работает в частотных диапазонах 430, 868 или 900 МГц. Более подробную информацию можно найти на странице RNode.
- **Хост-устройство**
  Любое компьютерное устройство под управлением Linux и Python. Рекомендуется использовать Raspberry Pi с ОС на базе Debian.
- **Программный стек**
  Самая последняя выпущенная реализация Reticulum на Python, работающая на операционной системе на базе Debian.

Во избежание путаницы очень важно отметить, что эталонное интерфейсное устройство не использует стандарт LoRaWAN, а применяет пользовательский MAC-уровень поверх обычной модуляции LoRa! Таким образом, вам понадобится обычный радиомодуль LoRa, подключенный к контроллеру с корректной прошивкой. Полная информация о том, как получить или создать такое устройство, доступна на странице RNode.

При текущей эталонной настройке подключиться к сети Reticulum можно примерно за 100$, даже если у вас нет аппаратного обеспечения и вам нужно приобрести все компоненты.

Эта эталонная настройка, конечно, является лишь рекомендацией для лёгкого старта, и вам следует адаптировать её к своим конкретным потребностям или к имеющемуся оборудованию.

4.6 Особенности протокола

В этой главе будет подробно описана специфическая для протокола информация, которая важна для реализации Reticulum, но не критична для общего понимания принципов работы протокола. К ней следует относиться скорее как к справочнику, чем как к обязательному чтению.

4.6.1 Приоритизация пакетов

В настоящее время Reticulum полностью не зависит от приоритетов в отношении общего трафика. Весь трафик обрабатывается в порядке поступления. Повторная передача объявлений обрабатывается в соответствии со временем повторной передачи и приоритетами, описанными ранее в этой главе.

4.6.2 Коды доступа к интерфейсам

Reticulum может создавать именованные виртуальные сети и сети, доступ к которым возможен только при знании предварительно согласованной парольной фразы. Конфигурация этого подробно описана в разделе *Common Interface Options*. Для реализации этих функций Reticulum использует концепцию кодов доступа к интерфейсу, которые рассчитываются и проверяются для каждого пакета.

Интерфейс с именованной виртуальной сетью или включенной аутентификацией по кодовой фразе будет использовать общую идентификацию подписи Ed25519 и для каждого исходящего пакета генерировать подпись всего пакета. Эта подпись затем вставляется в пакет в качестве кода доступа к интерфейсу (Interface Access Code) перед передачей. В зависимости от скорости и возможностей интерфейса, IFAC может быть полной 512-битной подписью Ed25519 или усеченной версией. Сконфигурированную длину IFAC можно проверить для всех интерфейсов с помощью утилиты `rnstatus`.

При получении интерфейс проверит, соответствует ли подпись ожидаемому значению, и отбросит пакет, если это не так. Это гарантирует, что только пакеты, отправленные с правильными параметрами именования и/или кодовой фразы, будут допущены в сеть.

4.6.3 Формат передачи данных

```
== Формат передачи данных Reticulum ======

Пакет Reticulum состоит из следующих полей:

[ЗАГОЛОВОК 2 байта] [АДРЕСА 16/32 байта] [КОНТЕКСТ 1 байт] [ДАННЫЕ 0-465 байт]

* Поле HEADER имеет длину 2 байта.
  * Байт 1: [Флаг IFAC], [Тип заголовка], [Флаг контекста], [Тип распространения], [Тип назначения] и [Тип пакета]
  * Байт 2: Количество переходов

* Поле кода доступа к интерфейсу, если установлен флаг IFAC.
  * Длина кода доступа к интерфейсу может варьироваться
    от 1 до 64 байт в соответствии с возможностями и конфигурацией физического интерфейса.

* Поле ADDRESSES содержит 1 или 2 адреса.
  * Каждый адрес имеет длину 16 байт.
  * Флаг типа заголовка в поле HEADER определяет, содержит
    ли поле ADDRESSES 1 или 2 адреса.
  * Адреса представляют собой хеши SHA-256, усеченные до 16 байт.

* Поле CONTEXT имеет размер 1 байт.
  * Оно используется Reticulum для определения контекста пакета.

* Поле DATA имеет размер от 0 до 465 байт.
  * Оно содержит полезную нагрузку данных пакета.

Флаг IFAC
-----------------
открыть             0 Пакет для общедоступного интерфейса
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
аутентифицированный   1   Аутентификация интерфейса включена в пакет.


Типы заголовков
-----------------
тип 1               0 Двухбайтовый заголовок, одно 16-байтовое адресное поле
тип 2               1   Двухбайтовый заголовок, два 16-байтовых адресных поля


Флаг контекста
-----------------
не установлен       0 Флаг контекста используется для различных типов
установлен          1   сигнализации, в зависимости от контекста пакета.


Типы распространения
-----------------
широковещательный       0
транспортный    1


Типы назначения
-----------------
единичный       00
группа          01
обычный         10
связь           11


Типы пакетов
-----------------
данные          00
анонс           01
запрос связи  10
подтверждение   11


+- Пример пакета -+

  ПОЛЕ ЗАГОЛОВКА          ПОЛЯ НАЗНАЧЕНИЯ               ПОЛЕ КОНТЕКСТНЫХ ДАННЫХ
 _______|_______    _________________|_________________    ________|______    __|_
|               | |                                   | |               | |    |
01010000 00000100 [HASH1, 16 bytes] [HASH2, 16 bytes] [CONTEXT, 1 byte] [DATA]
|| | | |     |
|| | | |     +-- Hops              = 4
|| | | +------- -Тип пакета        = ДАННЫЕ
|| | +-------Тип назначения = ОДИНОЧНЫЙ
|| +-------Тип распространения = ТРАНСПОРТ
|+-----------Тип заголовка        = HEADER_2 (двухбайтовый заголовок, два поля адреса)
+-------------- ----Коды доступа = DISABLED
```

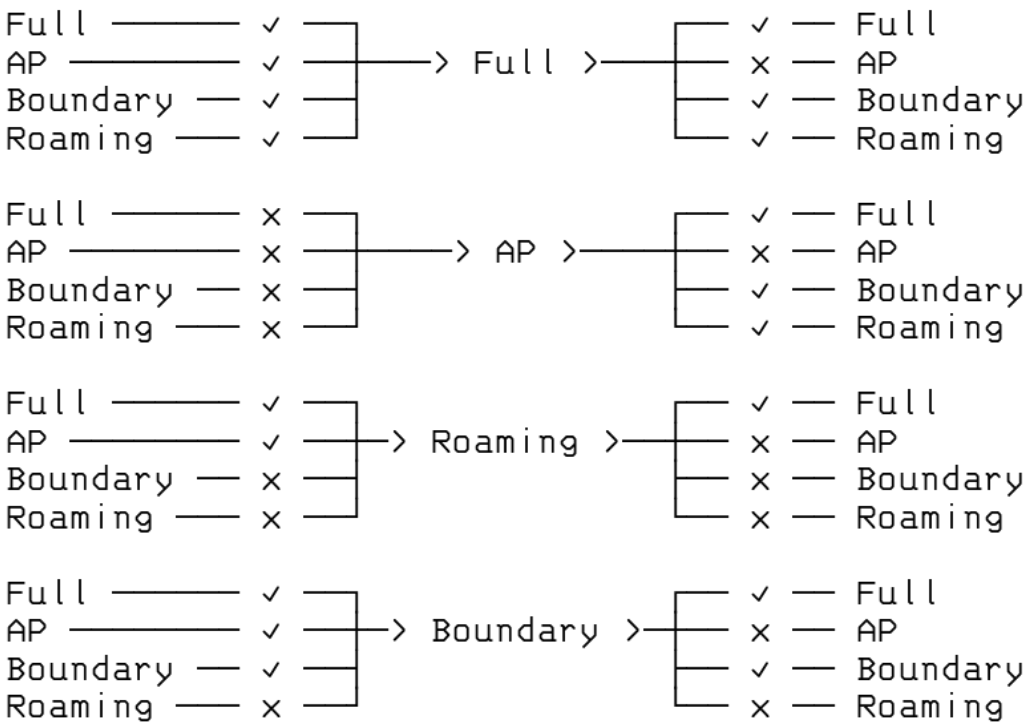
(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
+- Пример пакета -+

    ПОЛЕ ЗАГОЛОВКА ПОЛЕ НАЗНАЧЕНИЯ ПОЛЕ КОНТЕКСТА ПОЛЕ ДАННЫХ
 _______|_______   _______|_______   ________|______   __|_
|               | |               | |               | |  |000
00000 00000111 [HASH1, 16 байт] [CONTEXT, 1 байт] [DATA] || | | |
 | || | | +--
Хопы                              = 7
|| | | +------- -Тип пакета      = ДАННЫЕ
|| | +-------Тип назначения = ОДИНОЧНЫЙ
|| +--------Тип распространения = BROADCAST
|+-----------Тип заголовка       = HEADER_1 (двухбайтовый заголовок, одно поле адреса)
+-------------- ----Коды доступа = DISABLED


+- Пример пакета -+

    ПОЛЕ ЗАГОЛОВКА ПОЛЕ IFAC ПОЛЕ НАЗНАЧЕНИЯ ПОЛЕ КОНТЕКСТА ПОЛЕ ДАННЫХ
 _______|_______   ______|______   _______|_______   ________|______   __|_
|               | |              | |               | |               | |  |100
00000 00000111 [IFAC, N байт] [HASH1, 16 байт] [CONTEXT, 1 байт] [DATA] || | | |  |
|| | | +--
Хопы                              = 7
|| | | +------- -Тип пакета      = ДАННЫЕ
|| | +-------Тип назначения = ОДИНОЧНЫЙ
|| +--------Тип распространения = BROADCAST
|+-----------Тип заголовка       = HEADER_1 (двухбайтовый заголовок, одно поле адреса)
+-------------- ----Коды доступа = ENABLED


Примеры размеров различных типов пакетов
----------------------------------------

В следующей таблице приведены примеры размеров различ-
ных типов пакетов. Указанный размер является полным раз-
мером в сети, учитывающим все поля, включая заголовки,
но исключая любые коды доступа к интерфейсу.

- Запрос пути   :  51 байт
- Объявление        :      167 байт
- Запрос на соединение  :  83 байта
- Подтверждение соединения      :      115 байт
- Пакет RTT соединения :  99 байт
- Поддержание соединения  :  20 байт
```

4.6.4 Правила распространения объявлений

В следующей таблице проиллюстрированы правила автоматического распространения объявлений от одного типа интерфейса к другому, для всех возможных комбинаций. Для целей распространения объявлений режимы *Full* и *Gateway* идентичны.



См. раздел*Режимы интерфейса*для концептуального обзора различных режимов интерфейса и способов их настройки.

4.6.5 Криптографические примитивы

Reticulum использует простой набор эффективных, надёжных и хорошо протестированных криптографических примитивов с широко доступными реализациями, которые могут быть использованы как на процессорах общего назначения, так и на микроконтроллерах.

Одним из основных соображений при выборе данного набора примитивов является то, что они могут быть реализованы безопасно с относительно небольшим количеством подводных камней практически на всех современных вычислительных платформах.

Перечисленные здесь примитивы **являются авторитетными** . Всё, что претендует на название Reticulum, но не использует эти точные примитивы, не является Reticulum и, возможно, представляет собой намеренно скомпрометированный или ослабленный клон. Используемые примитивы:

- Ed25519 для подписей
- X25519 для обмена ключами ECDH
- HKDF для вывода ключей
- Зашифрованные токены основаны на спецификации Fernet.
  - Эфемерные ключи, полученные в результате обмена ключами ECDH на Curve25519.
  - AES-256 в режиме CBC с дополнением PKCS7
  - HMAC с использованием SHA256 для аутентификации сообщений
  - IV должны генерироваться через`os.urandom()`или более надёжным способом.
  - Отсутствуют поля метаданных версии Fernet и временной метки.

- SHA-256
- SHA-512

В конфигурации установки по умолчанию примитивы `X25519`, `Ed25519` и `AES-256-CBC` предоставляются OpenSSL (через пакет PyCA/cryptography). Хеш-функции `SHA-256` и `SHA-512` предоставляются стандартной библиотекой Python hashlib. Примитивы `HKDF`, `HMAC`, `Token` и функция дополнения `PKCS7` всегда предоставляются следующими внутренними реализациями:

- `RNS/Cryptography/HKDF.py`
- `RNS/Cryptography/HMAC.py`
- `RNS/Cryptography/Token.py`
- `RNS/Cryptography/PKCS7.py`

Reticulum также включает полную реализацию всех необходимых примитивов на чистом Python. Если OpenSSL и PyCA недоступны в системе при запуске Reticulum, Reticulum будет использовать внутренние примитивы на чистом Python. Тривиальным следствием этого является производительность, при этом бэкенд OpenSSL работает намноготбыстрее. Однако наиболее важным следствием является потенциальная потеря безопасности при использовании примитивов, которые не подвергались такому же тщательному изучению, тестированию и проверке, как примитивы из OpenSSL.

Предупреждение

Если вы хотите использовать внутренние примитивы на чистом Python, настоятельно **рекомендуется** хорошо понимать связанные с этим риски и принять обоснованное решение о том, приемлемы ли они для вас.

ГЛАВА

ПЯТЬ

КОММУНИКАЦИОННОЕ ОБОРУДОВАНИЕ

Одним из действительно ценных аспектов Reticulum является возможность его использования практически с любым мыслимым видом средств связи. Типы интерфейсов, доступные для настройки в Reticulum, достаточно гибки, чтобы охватывать использование большинства доступного проводного и беспроводного коммуникационного оборудования — от пакетных радиомодемов десятилетней давности до современных миллиметровых систем обратной связи.

Если у вас уже есть или вы эксплуатируете какое-либо коммуникационное оборудование, очень велика вероятность того, что оно будет работать с Reticulum «из коробки». В противном случае можно легко обеспечить необходимое взаимодействие, используя, например, *PipeInterface* или *TCPClientInterface* в сочетании с таким кодом, как TCP KISS Server, просто установив его.

Также очень легко писать и загружать *пользовательские модули интерфейса* в Reticulum, что позволяет вам общаться практически со всем, о чем вы только можете подумать.

Хотя такая широкая поддержка и гибкость очень полезны, обилие опций иногда может затруднить понимание того, с чего начать, особенно когда вы начинаете с нуля.

Эта глава изложит несколько различных разумных начальных путей для запуска реальных функциональных беспроводных коммуникаций с минимальными затратами и усилиями. Будут рассмотрены две основные категории устройств: *RNodes* и *радиостанции на основе WiFi* . Кроме того, будут кратко описаны другие распространённые вари-

Знание того, как использовать всего несколько различных типов аппаратного обеспечения, позволит с небольшими усилиями построить широкий спектр полезных сетей.

5.1 Комбинирование типов оборудования

При проектировании и строительстве сети полезно комбинировать различные типы каналов и оборудования. Одним из полезных шаблонов проектирования является использование высокопроизводительных каналов точка-точка на основе WiFi или миллиметровых радиостанций (с высоконаправленными антеннами) для магистрали сети и использование RNodes на основе LoRa для покрытия больших территорий связью для клиентских устройств.

5.2 RNode

Надёжные и универсальные системы приёмопередатчиков цифровой радиосвязи дальнего радиуса действия обычно либо очень дороги, либо сложны в настройке и эксплуатации, либо труднодоступны, либо энергоёмки, либо всё вышеперечисленное одновременно. В попытке облегчить эту ситуацию была разработана приёмопередающая система *RNode* . Важно отметить, что RNode — это не конкретное устройство от одного производителя, а *открытая платформа* , которую каждый может использовать для создания совместимых цифровых приемопередатчиков, соответствующих его потребностям и конкретным ситуациям.

RNode — это универсальное, совместимое, маломощное и дальнобойное, надёжное, открытое и гибкое радиокоммуникационное устройство. В зависимости от своих компонентов, он может работать на множестве различных частотных диапазонов и использовать множество различных схем модуляции, но чаще всего, и для целей этой главы, мы ограничим обсуждение RNodes, использующих модуляцию *LoRa* в обычных ISM-диапазонах.

Избегайте путаницы! RNodes могут использовать LoRa как *модуляцию физического уровня* , но они не используют и не имеют ничего общего с протоколом и стандартом *LoRaWAN* , обычно применяемым для централизованно управляемых IoT-устройств. RNodes используют *чистую модуляцию LoRa* , без какого-либо дополнительного протокольного оверхеда. Вся функциональность высокоуровневых протоколов обрабатывается непосредственно Reticulum.

5.2.1 Создание RNodes

RNode был разработан как система, которую легко реплицировать во времени и пространстве. Вы можете собрать функционирующий трансивер, используя общедоступные компоненты и несколько программных инструментов с открытым исходным кодом. Хотя вы можете проектировать и создавать RNodes полностью с нуля, по вашим точным желаемым спецификациям, в этой главе будет объяснен самый простой подход к созданию RNodes: использование общих плат разработки LoRa. Этот подход можно свести к двум простым шагам

1. Получить одну или несколько*поддерживаемых плат разработки*
2. Установить прошивку RNode с помощью*автоматического установщика*

После установки и настройки прошивки установочным скриптом она готова к использованию с любым программным обеспечением, поддерживающим RNodes, включая Reticulum. Устройство можно использовать с Reticulum, добавив *RNodeInterface* в конфигурацию.

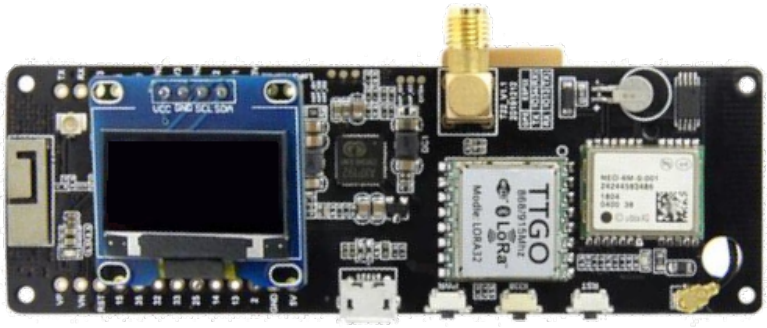
5.2.2 Поддерживаемые платы и устройства

Для создания одного или нескольких RNodes вам потребуется приобрести поддерживаемые платы разработки или готовые устройства. Следующие платы и устройства поддерживаются автоинсталлятором.



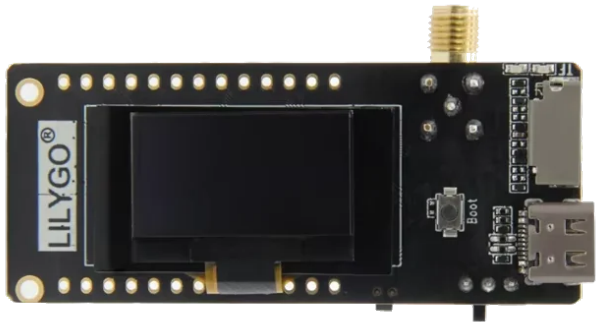
LilyGO T-Beam Supreme

- **Приемопередающая ИС**Semtech SX1262 или SX1268
- **Платформа устройства** ESP32
- **Производитель** LilyGO



LilyGO T-Beam

- **Приемопередающая ИС**Semtech SX1262, SX1268, SX1276 или SX1278
- **Платформа устройства** ESP32
- **Производитель** LilyGO



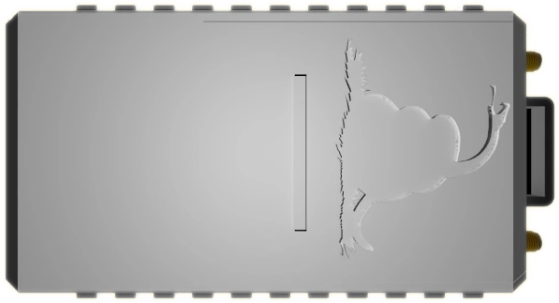
LilyGO T3S3

- **Приемопередающая ИС**Semtech SX1262, SX1268, SX1276 или SX1278
- **Платформа устройства** ESP32
- **Производитель** LilyGO



Платы на базе RAK4631

- **Приемопередающая ИС**Semtech SX1262 или SX1268
- **Платформа устройства** nRF52
- **Производитель**RAK Wireless



OpenCom XL

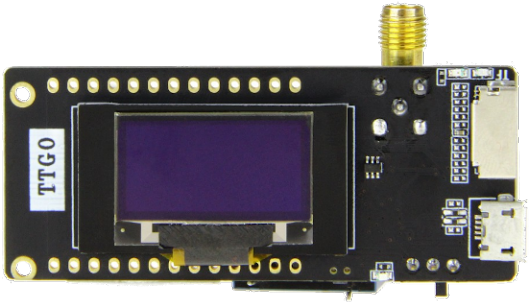
- **Приемопередающие ИС**Semtech SX1262 и SX1280 (двойной приемопередатчик)
- **Платформа устройства** nRF52
- **Производитель**RAK Wireless



Unsigned RNode v2.x

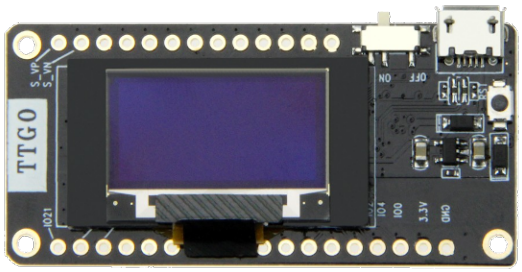
- **Приемопередающая ИС**Semtech SX1276 или SX1278
- **Платформа устройства** ESP32

- **Производитель**unsigned.io



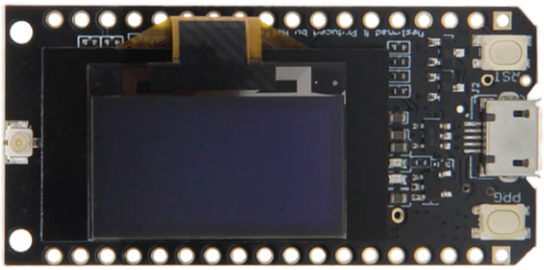
LilyGO LoRa32 v2.1

- **Приемопередающая ИС**Semtech SX1276 или SX1278
- **Платформа устройства** ESP32
- **Производитель** LilyGO



LilyGO LoRa32 v2.0

- **Приемопередающая ИС**Semtech SX1276 или SX1278
- **Платформа устройства** ESP32
- **Производитель** LilyGO



LilyGO LoRa32 v1.0

- **Приемопередающая ИС**Semtech SX1276 или SX1278
- **Платформа устройства** ESP32
- **Производитель** LilyGO



LilyGO T-Deck

- **Приемопередающая ИС**Semtech SX1262 или SX1268
- **Платформа устройства** ESP32
- **Производитель** LilyGO



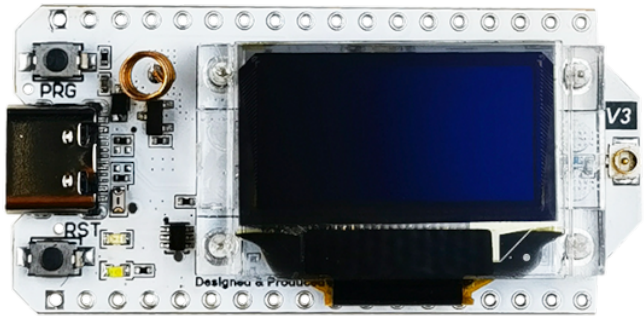
LilyGO T-Echo

- **Приемопередающая ИС**Semtech SX1262 или SX1268
- **Платформа устройства** nRF52
- **Производитель** LilyGO



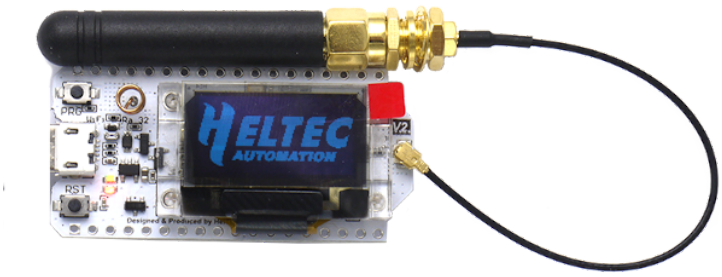
Heltec T114

- **Приемопередающая ИС**Semtech SX1262 или SX1268
- **Платформа устройства** nRF52
- **Производитель**Heltec Automation



Heltec LoRa32 v3.0

- **Приемопередающая ИС**Semtech SX1262 или SX1268
- **Платформа устройства** ESP32
- **Производитель**Heltec Automation



Heltec LoRa32 v2.0

- **Приемопередающая ИС**Semtech SX1276 или SX1278
- **Платформа устройства** ESP32
- **Производитель**Heltec Automation

5.2.3 Установка

После того как вы приобрели совместимые платы, вы можете установить прошивку RNode с помощью утилиты настройки RNode. Если вы установили Reticulum в вашей системе, программа `rnodeconf` уже будет доступна. Если нет, убедитесь, что `Python3` и `pip` установлены в вашей системе, а затем установите Reticulum с помощью `pip` :

```
pip install rns
```

После завершения установки пришло время начать устанавливать прошивку на ваши устройства. Запустите `rnodeconf` в режиме автоустановки следующим образом:

```
rnodeconf --autoinstall
```

Утилита проведет вас через процесс установки, задавая ряд вопросов о вашем оборудовании. Просто следуйте руководству, и утилита автоматически установит и настроит ваши устройства.

5.2.4 Использование с Reticulum

После установки и настройки устройств вы можете использовать их с Reticulum, добавив соответствующий раздел интерфейса в файл конфигурации Reticulum. В конфигурации вы можете указать все параметры интерфейса, такие как последовательный порт и параметры эфира.

5.3 Аппаратное обеспечение на основе WiFi

С Reticulum можно использовать все виды аппаратного обеспечения на основе WiFi как ближнего, так и дальнего радиуса действия. Любое аппаратное обеспечение, которое полностью поддерживает мостовой Ethernet через интерфейс WiFi, будет работать с *AutoInterface* в Reticulum. Большинство устройств будут вести себя так по умолчанию или позволят это через параметры конфигурации

Это означает, что вы можете просто настроить физические каналы устройств на основе WiFi и начать обмениваться данными через них с помощью Reticulum. Нет необходимости включать какую-либо IP-инфраструктуру, такую как DHCP-серверы, DNS или аналогичные, если доступен хотя бы Ethernet, и пакеты прозрачно передаются через физические устройства на основе WiFi.

Ниже приведен список примеров Wi-Fi (и аналогичных) радиостанций, которые хорошо подходят для высокопроизводительных соединений Reticulum на больших расстояниях:

- Радиостанции Ubiquiti airMAX

- Радиостанции Ubiquiti LTU
- Радиостанции MikroTik

Этот список ни в коем случае не является исчерпывающим и служит лишь несколькими примерами относительно недорогого радиооборудования, обеспечивающего большую дальность действия и высокую пропускную способность для сетей Reticulum. Как и во всех других случаях, Reticulum также может сосуществовать с IP-сетями, работающими одновременно на таких устройствах.

5.4 Аппаратное обеспечение на основе Ethernet

Reticulum может работать на любом типе оборудования, которое может обеспечить коммутируемую среду на основе Ethernet. Это означает, что Reticulum может использовать все: от простого Ethernet-коммутатора до оптоволоконных систем и радиомодемов с интерфейсами Ethernet.

Среда Ethernet не нуждается в какой-либо IP-инфраструктуре, такой как DHCP-серверы или маршрутизация, но если такая инфраструктура существует, Reticulum будет просто сосуществовать с ней.

Для использования Reticulum в средах на основе Ethernet обычно достаточно использовать прилагаемый *AutoInterface* . Этот интерфейс также работает со всеми видами виртуальных сетевых адаптеров, таких как устройства `tun` и `tap` в Linux.

5.5 Последовательные линии и устройства

Использование Reticulum через любую необработанную последовательную линию также возможно с помощью *SerialInterface* . Этот тип интерфейса также полезен для использования Reticulum с коммуникационным оборудованием, которое предоставляет интерфейс последовательного порта.

5.6 Пакетные радиомодемы

Любой пакетный радиомодем, предоставляющий стандартный интерфейс KISS через USB, последовательный порт или TCP, может использоваться с Reticulum. Это включает виртуальные программные модемы, такие как FreeDV TNC и Dire Wolf.

ГЛАВА

ШЕСТЬ

НАСТРОЙКА ИНТЕРФЕЙСОВ

Reticulum поддерживает использование многих видов устройств в качестве сетевых интерфейсов и позволяет смешивать и сопоставлять их любым способом по вашему выбору. Количество различных сетевых топологий, которые вы можете создать с помощью Reticulum, более или менее бесконечно, но общее для всех них то, что вам нужно будет определить один или несколько интерфейсов для использования Reticulum.

В следующих разделах описаны интерфейсы, доступные в Reticulum, и приведены примеры конфигураций для соответствующих типов интерфейсов.

Для получения общего обзора того, как сети могут быть сформированы с использованием различных типов интерфейсов, ознакомьтесь с главой «Построение сетей» в этом руководстве.

6.1 Пользовательские интерфейсы

Помимо встроенных типов интерфейсов, Reticulum полностью расширяем с помощью пользовательских интерфейсов, предоставляемых пользователями или сообществом, и создание пользовательских модулей интерфейсов несложно. Пожалуйста, ознакомьтесь с примером пользовательского интерфейса для получения базового кода интерфейса, на основе которого можно строить.

6.2 Автоматический интерфейс

`AutoInterface` обеспечивает связь с другими обнаруживаемыми узлами Reticulum через любые локальные среды Ethernet или Wi-Fi. Несмотря на то, что он использует IPv6 для обнаружения пиров и UDP для передачи пакетов, ему не требуется никакая функциональная IP-инфраструктура, такая как маршрутизаторы или DHCP-серверы, в вашей физической сети.

Предупреждение

Если на вашем компьютере запущено программное обеспечение брандмауэра, оно может блокировать трафик, необходимый для работы `AutoInterface`. В этом случае вам придется разрешить UDP-трафик на портах `29716` и `42671`.

Пока между пирами присутствует хотя бы какая-то среда коммутации (проводной коммутатор, концентратор, точка доступа Wi-Fi или аналогичное устройство, либо просто два устройства, подключенные напрямую Ethernet-кабелем), он будет работать без какой-либо настройки, установки или промежуточных устройств.

Для работы обнаружения пиров `AutoInterface` также требуется поддержка link-local IPv6 в вашей системе, что должно быть по умолчанию во всех современных операционных системах, как настольных, так и мобильных.

Примечание

Почти все современное оборудование Ethernet и Wi-Fi будет работать без какой-либо настройки или установки с `AutoInterface` , но небольшая часть устройств включает опции, которые ограничивают связь между устройствами по умолча-

что приводит к блокировке обнаружения пиров `AutoInterface`. Эта проблема чаще всего встречается на очень дешевых Wi-Fi-маршрутизаторах, поставляемых провайдерами, и иногда ее можно отключить в конфигурации маршрутизатора.

```
# Этот пример демонстрирует минимальную настройку
# автоматического интерфейса. Это позволит обмениваться данными
# со всеми другими доступными устройствами на всех
# используемых физических устройствах на базе Ethernet,
# которые доступны в системе.
[[Default Interface]]
  type = AutoInterface
  enabled = yes

# Этот пример демонстрирует более конкретно настроен-
ный автоматический интерфейс, который использует
только определённые физические интерфейсы и имеет ряд
других установленных параметров конфигурации.
[[Default Interface]]
  type = AutoInterface
  enabled = yes

  # Вы можете создать несколько изолированных сетей Reticulum
  # в одной и той же физической локальной сети, указав
  # различные идентификаторы групп.
  group_id = reticulum

  # Вы также можете выбрать тип многоадресного адреса:
  # временный (по умолчанию, временный многоадресный адрес)
  # или постоянный (постоянный многоадресный адрес)
  multicast_address_type = permanent

  # Вы также можете выбрать конкретные сете-
  вые устройства ядра для использования.
  devices = wlan0,eth1

  # Или позволить AutoInterface использовать все подходящие
  устройства, за исключением списка игнорируемых.
  ignored_devices = tun0,eth0
```

Если вы подключены к Интернету по IPv6 и ваш провайдер маршрутизирует многоадресную рассылку IPv6, вы потенциально можете настроить автоинтерфейс для глобального автоматического обнаружения других узлов Reticulum в пределах выбранного вами Group ID. Вы можете указать область обнаружения, установив ее в одно из следующих значений: `link`, `admin`, `site`, `organisation` или `global`.

```
[[Default Interface]]
  type = AutoInterface
  enabled = yes

  # Настроить глобальное обнаружение

  group_id = custom_network_name
  discovery_scope = global

  # Другие параметры конфигурации
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  discovery_port = 48555
  data_port = 49555
```

6.3 Backbone Interface

Интерфейс Backbone — это очень быстрый и ресурсоэффективный тип интерфейса, в основном предназначенный для соединения экземпляров Reticulum через множество различных типов сред. Он использует бэкэнд ввода-вывода на основе событий ядра и может обрабатывать тысячи интерфейсов и/или клиентов с относительно низким использованием системных ресурсов. **Этот тип интерфейса в настоящее время поддерживается только на Linux и Android .**

Примечание

Интерфейс Backbone полностью совместим с типами `TCPServerInterface` и `TCPClientInterface` , и они могут использоваться взаимозаменяемо и соединяться друг с другом. В системах, поддерживающих `BackboneInterface` , обычно рекомендуется использовать его, если вам не требуются специфические опции или функции, предоставляемые интерфейсами TCP-сервера и клиента.

Хотя цель состоит в поддержке *all* типов сокетов и устройств ввода-вывода, предоставляемых базовой операционной системой, первоначальный выпуск обеспечивает поддержку только TCP-соединений по IPv4 и IPv6.

Для всех типов соединений через `BackboneInterface` Reticulum будет корректно обрабатывать прерывания, потерю связи и временные подключения.

6.3.1 Слушатели

Следующие примеры иллюстрируют различные способы настройки слушателей`BackboneInterface` .

```
# Этот пример демонстрирует магистральный интерфейс,
# который прослушивает входящие соединения на
# указанный IP-адрес и номер порта.
[[Backbone Listener]]
  type = BackboneInterface
  enabled = yes
  listen_on = 0.0.0.0
  port = 4242

# В качестве альтернативы можно привязаться к определенному IP-адресу
[[Backbone Listener]]
  type = BackboneInterface
  enabled = yes
  listen_on = 10.0.0.88
  port = 4242

# Или конкретное сетевое устройство
[[Backbone Listener]]
  type = BackboneInterface
  enabled = yes
  device = eth0
  port = 4242
```

Если вы используете интерфейс на устройстве, для которого доступны как IPv4, так и IPv6-адреса, вы можете использовать опцию `prefer_ipv6` для привязки к IPv6-адресу:

```
# Этот пример демонстрирует магистральный интерфейс,
# прослушивание на IPv6-адресе указанного
# сетевого устройства ядра.
[[Backbone Listener]]
  type = BackboneInterface
  enabled = yes
  prefer_ipv6 = yes
  device = eth0
  port = 4242
```

Чтобы использовать `BackboneInterface` через Yggdrasil, вы можете просто указать устройство Yggdrasil `tun` и порт прослушивания, например:

```
# Этот пример демонстрирует магистральный интерфейс,
# прослушивание подключений через Yggdrasil.
[[Yggdrasil Backbone Interface]]
  type = BackboneInterface
  enabled = yes
  device = tun0
  port = 4343
```

6.3.2 Подключение удаленных устройств

Следующие примеры иллюстрируют различные способы подключения к удаленным `BackboneInterface` прослушивателям. Как отмечалось выше, интерфейсы `BackboneInterface` также могут подключаться к удаленным `TCPServerInterface` , и, таким образом, эти типы интерфейсов могут использоваться взаимозаменяемо.

```
# Здесь'пример магистрального интерфейса, который
# подключается к удаленному слушателю.
[[Backbone Remote]]
  type = BackboneInterface
  enabled = yes
  remote = amsterdam.connect.reticulum.network
  target_port = 4251
```

Для подключения к удаленным узлам через Yggdrasil просто укажите целевой IPv6-адрес и порт Yggdrasil, как показано ниже:

```
[[Yggdrasil Remote]]
    type = BackboneInterface
    enabled = yes
    target_host = 201:5d78:af73:5caf:a4de:a79f:3278:71e5
    target_port = 4343
```

6.4 Интерфейс TCP-сервера

Интерфейс TCP-сервера подходит для подключения других пиров через Интернет или частные сети IPv4 и IPv6. Когда интерфейс TCP-сервера настроен, другие пиры Reticulum могут подключаться к нему с помощью интерфейса TCP-клиента.

```
# Этот пример демонстрирует интерфейс TCP-сервера.
# Он будет прослушивать входящие соединения на всех IP-адресах
# интерфейсов на порту 4242.
[[TCP Server Interface]]
  type = TCPServerInterface
  enabled = yes
  listen_ip = 0.0.0.0
  listen_port = 4242

# В качестве альтернативы можно привязаться к определенному IP-адресу
[[TCP Server Interface]]
  type = TCPServerInterface
  enabled = yes
  listen_ip = 10.0.0.88
  listen_port = 4242

# Или конкретное сетевое устройство
[[TCP Server Interface]]
  type = TCPServerInterface
  enabled = yes
  device = eth0
  listen_port = 4242
```

Если вы используете интерфейс на устройстве, для которого доступны как IPv4, так и IPv6-адреса, вы можете использовать опцию `prefer_ipv6` для привязки к IPv6-адресу:

```
# Этот пример демонстрирует интерфейс TCP-сервера.
# Он будет прослушивать входящие соединения по
# указанному IP-адресу и номеру порта.

[[TCP Server Interface]]
  type = TCPServerInterface
  enabled = yes
  prefer_ipv6 = True
  device = eth0
  port = 4242
```

Чтобы использовать интерфейс TCP-сервера через Yggdrasil, вы можете просто указать устройство Yggdrasil `tun` и порт прослушивания, например так:

```
[[Yggdrasil TCP Server Interface]]
  type = TCPServerInterface
  enabled = yes
  device = tun0
  listen_port = 4343
```

Примечание

Интерфейсы TCP поддерживают туннелирование через I2P, но для надёжной работы необходимо использовать опцию i2p_tunneled:

```
[[TCP-сервер в I2P]]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  type = TCPServerInterface
  enabled = yes
  listen_ip = 127.0.0.1
  listen_port = 5001
  i2p_tunneled = yes
```

Почти во всех случаях проще использовать выделенный `I2PInterface` , но для полного контроля и использования маршрутизаторов I2P, работающих на внешних системах, этот вариант также существует

6.5 Интерфейс TCP-клиента

Для подключения к интерфейсу TCP-сервера можно использовать интерфейс TCP-клиента. Множество интерфейсов TCP-клиентов от разных пиров могут одновременно подключаться к одному и тому же интерфейсу TCP-сервера.

Типы TCP-интерфейсов также могут выдерживать прерывания на канальном уровне IP. Это означает, что Reticulum будет корректно обрабатывать IP-каналы, которые поднимаются и опускаются, и восстанавливать соединение после сбоя, как только другая сторона TCP-интерфейса снова появится.

```
# Вот ' пример интерфейса TCP-клиента. # target_host
может быть именем хоста, IPv4- или IPv6-адресом.
[[TCP Client Interface]]
  type = TCPClientInterface
  enabled = yes
  target_host = 127.0.0.1
  target_port = 4242
```

Чтобы использовать клиентский интерфейс TCP через Yggdrasil, просто укажите целевой IPv6-адрес Yggdrasil и порт, как показано ниже:

```
[[Yggdrasil TCP Client Interface]]
    type = TCPClientInterface
    enabled = yes
    target_host = 201:5d78:af73:5caf:a4de:a79f:3278:71e5
    target_port = 4343
```

Также возможно использовать этот тип интерфейса для подключения через другие программы или аппаратные устройства, которые предоставляют интерфейс KISS на TCP-порту, например, программные звуковые модемы. Для этого используйте опцию `kiss_framing` :

```
# Вот пример клиентского интерфейса TCP, который подключается
# к программному звуковому модему TNC через порт KISS over TCP.

[[TCP KISS Interface]]
  type = TCPClientInterface
  enabled = yes
  kiss_framing = True
  target_host = 127.0.0.1
  target_port = 8001
```

Внимание! Используйте опцию KISS-кадрирования только при подключении к внешним устройствам и программам, таким как звуковые модемы и аналогичные по TCP. При использовании `TCPClientInterface` совместно с `TCPServerInterface` никогда не включайте `kiss_framing` , так как это отключит внутренние механизмы надёжности и восстановления, которые значительно улучшают производительность по ненадёжным и прерывистым TCP-соедине-

Примечание

Интерфейсы TCP поддерживают туннелирование через I2P, но для надёжной работы необходимо использовать опцию i2p_tunneled:

```
[[TCP Client over I2P]]
    type = TCPClientInterface
    enabled = yes
    target_host = 127.0.0.1
    target_port = 5001
    i2p_tunneled = yes
```

6.6 UDP-интерфейс

UDP-интерфейс может быть полезен для связи по IP-сетям, как частным, так и в интернете. Он также может обеспечивать широковещательную связь по IP-сетям, предоставляя простой способ для подключения ко всем другим узлам в локальной сети.

Предупреждение

Использование широковещательного UDP-трафика влияет на производительность, особенно в Wi-Fi. Если ваша цель состоит в том, чтобы просто обеспечить лёгкую связь со всеми узлами в вашем локальном широковещательном домене Ethernet, то *Auto Interface* работает гораздо лучше и даже проще в использовании.

```
# Этот пример обеспечивает связь с другими # ло-
кальными узлами Reticulum по UDP.

[[UDP Interface]]
  type = UDPInterface
  enabled = yes

  listen_ip = 0.0.0.0
  listen_port = 4242
  forward_ip = 255.255.255.255
  forward_port = 4242

  # Вышеуказанная конфигурация позволит осуществлять связь
  # в пределах локальных широковещательных доменов всех локальных
  # IP-интерфейсов.

  # Вместо указания listen_ip, listen_port,
  # forward_ip и forward_port, вы также можете привязаться
  # к определенному сетевому устройству, как показано ниже.

  # device = eth0
  # port = 4242

  # Предполагая, что устройство eth0 имеет адрес
  # 10.55.0.72/24, вышеуказанная конфигурация будет
  # эквивалентна следующей ручной настройке.
  # Обратите внимание, что мы и слушаем, и пересылаем на
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  # широковещательный адрес сетевых сегментов.

  # listen_ip = 10.55.0.255
  # listen_port = 4242
  # forward_ip = 10.55.0.255
  # forward_port = 4242

  # Конечно, вы также можете общаться только с
  # одним IP-адресом

  # listen_ip = 10.55.0.15
  # listen_port = 4242
  # forward_ip = 10.55.0.16
  # forward_port = 4242
```

6.7 Интерфейс I2P

Интерфейс I2P позволяет подключать экземпляры Reticulum через протокол Invisible Internet Protocol. Это может быть особенно полезно в случаях, когда вы хотите разместить глобально доступный экземпляр Reticulum, но не имеете доступа к публичным IP-адресам, имеете часто меняющийся IP-адрес или когда брандмауэры блокируют входящий трафик.

Используя интерфейс I2P, вы получите глобально доступный, переносимый и постоянный I2P-адрес, по которому можно будет связаться с вашим экземпляром Reticulum.

Чтобы использовать интерфейс I2P, на вашей системе должен быть запущен маршрутизатор I2P. Самый простой способ добиться этого — загрузить и установить последнюю версию пакета `i2pd`. Для получения более подробной информации об I2P см. веб-сайт geti2p.net.

Когда маршрутизатор I2P работает в вашей системе, вы можете просто добавить интерфейс I2P в Reticulum:

```
[[I2P]]
  type = I2PInterface
  enabled = yes
  connectable = yes
```

При первом запуске Reticulum сгенерирует новый I2P-адрес для интерфейса и начнет прослушивать входящий трафик. В первый раз это может занять некоторое время, особенно если ваш маршрутизатор I2P также был только что запущен и еще не очень хорошо подключен к сети I2P. Когда все будет готово, вы должны увидеть I2P base32-адрес, записанный в файл журнала. Вы также можете проверить статус интерфейса с помощью утилиты `rnstatus`.

Чтобы подключиться к другим экземплярам Reticulum через I2P, просто добавьте список I2P base32-адресов, разделенных запятыми, к опции `peers` интерфейса:

```
[[I2P]]
  type = I2PInterface
  enabled = yes
  connectable = yes
  peers = 5urvjicpzi7q3ybztsef4i5ow2aq4soktfj7zedz53s47r54jnqq.b32.i2p
```

Установление I2P-соединений с желаемыми узлами может занимать от нескольких секунд до нескольких минут, поэтому Reticulum обрабатывает этот процесс в фоновом режиме и выводит соответствующие события в журнал.

Примечание

Хотя I2P-интерфейс является самым простым способом использования Reticulum через I2P, также возможно вручную туннелировать интерфейсы TCP-сервера и клиента через I2P. Это может быть полезно в ситуациях, когда требуется больший контроль, но требует ручной настройки туннеля через конфигурацию демона I2P.

Важно отметить, что эти два метода *взаимозаменяемо совместимы* . Например, вы можете использовать I2PInterface для подключения к TCPServerInterface, который был вручную туннелирован через I2P. Это обеспечивает высокую степень гибкости в настройке сети, сохраняя при этом простоту использования в более простых сценариях.

6.8 Интерфейс RNode LoRa

Для использования Reticulum через LoRa можно использовать интерфейс RNode, который предоставляет полный контроль над параметрами LoRa.

Предупреждение

Радиочастотный спектр является законодательно контролируемым ресурсом, и законодательство в разных странах мира значительно различается. Вы несете ответственность за ознакомление с любыми соответствующими нормативными актами для вашего местоположения и за принятие решений соответствующим образом.

```
# Вот'пример добавления интерфейса LoRa
# с использованием приемопередатчика LoRa RNode.

[[RNode LoRa Interface]]
  type = RNodeInterface

  # Включите интерфейс, если хотите его использовать!
  enabled = yes

  # Последовательный порт для устройства
  port = /dev/ttyUSB0

  # Также возможно использовать устройства BLE # вместо
  проводных последовательных портов. Целевой # RNode должен
  быть сопряжен с # хост-устройством перед подключением.
  Устройства BLE # могут быть подключены по имени, # BLE MAC
  -адресу или по любому доступному.


  # Подключиться к конкретному устройству по имени
  # port = ble://RNode 3B87

  # Или по BLE MAC-адресу
  # port = ble://F4:12:73:29:4E:89

  # Или подключиться к первому доступному,
  # сопряженное устройство
  # порт = ble://

  # Установите частоту на 867,2 МГц
  frequency = 867200000

  # Установите полосу пропускания LoRa на 125 КГц
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  bandwidth = 125000

  # Установите мощность TX на 7 dBm (5 мВт)
  txpower = 7

  # Выберите коэффициент расширения 8.
  Допустимый # диапазон от 7 до 12, где 7 #
  является самым быстрым, а 12 имеет # са-
  мый длинный диапазон.
  spreadingfactor = 8

  # Выберите скорость кодирования 5. Допустимый
  диапазон # от 5 до 8, где 5 # является самым
  быстрым, а 8 — самым длинным.
  codingrate = 5

  # Вы можете настроить RNode на отправку
  # идентификации по каналу с
  # установленным интервалом, настроив
  # следующие два параметра.

  # id_callsign = MYCALL-0
  # id_interval = 600

  # Для некоторых самодельных интерфейсов RNode
  # с малым объемом оперативной памяти, использование
  # управления потоком пакетов может быть полезным. По умолчанию
  # оно отключено.

  # flow_control = False

  # Можно ограничить использование эфирного
  времени RNode с помощью следующих двух
  параметров конфигурации.
  # Краткосрочное ограничение применяется в
  # окне приблизительно 15 секунд,
  # а долгосрочное ограничение действует
  # в течение скользящего 60-минутного окна. Оба
  # параметра указываются в процентах.

  # airtime_limit_long = 1.5
  # airtime_limit_short = 33
```

6.9 Многоинтерфейсный RNode

Для RNodes, поддерживающих несколько LoRa-трансиверов, многоинтерфейсный RNode может использоваться для индивидуальной настройки субинтерфейсов.

Предупреждение

Радиочастотный спектр является законодательно контролируемым ресурсом, и законодательство в разных странах мира значительно различается. Вы несете ответственность за ознакомление с любыми соответствующими нормативными актами для вашего местоположения и за принятие решений соответствующим образом.

```
# Здесь пример того, как добавить мультиинтерфейс RNode
# с использованием приемопередатчика LoRa RNode.

[[RNode Multi Interface]]
type = RNodeMultiInterface

# Включите интерфейс, если хотите его использовать!
enabled = yes

# Последовательный порт для устройства
port = /dev/ttyACM0

# Вы можете настроить RNode на отправку
# идентификации по каналу с
# установленным интервалом, настроив
# следующие два параметра.

# id_callsign = MYCALL-0
# id_interval = 600

  # Дополнительный интерфейс
  [[[High Datarate]]]
    # Дополнительные интерфейсы можно включать и выключать
    enabled = yes

    # Установите частоту 2.4ГГц
    frequency = 2400000000

    # Установите полосу пропускания LoRa на 1625 КГц
    bandwidth = 1625000

    # Установите мощность TX на 0 dBm (0.12 mW)
    txpower = 0

    # Виртуальный порт, только производитель
    # или человек, написавший конфигурацию платы,
    # может сказать вам, каким он будет для какого
    # интерфейса физического оборудования
    vport = 1

    # Выберите коэффициент расширения 5.
    Допустимый # диапазон от 5 до 12, где 5 #
    является самым быстрым, а 12 имеет # са-
    мый длинный диапазон.
    spreadingfactor = 5

    # Выберите скорость кодирования 5. Допустимый
    диапазон # от 5 до 8, где 5 # является самым
    быстрым, а 8 — самым длинным.
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    codingrate = 5

    # Можно ограничить использование эфирного
    времени RNode с помощью следующих двух
    параметров конфигурации.
    # Краткосрочное ограничение применяется в
    # окне приблизительно 15 секунд,
    # а долгосрочное ограничение действует
    # в течение скользящего 60-минутного окна. Оба
    # параметра указываются в процентах.

    # airtime_limit_long = 100
    # airtime_limit_short = 100

  [[[Низкая скорость передачи данных]]]
    # Дополнительные интерфейсы можно включать и выключать
    enabled = yes

    # Установите частоту на 865.6 МГц
    frequency = 865600000

    # Виртуальный порт, только производитель
    # или человек, написавший конфигурацию платы,
    # может сказать вам, каким он будет для какого
    # интерфейса физического оборудования
    vport = 0

    # Установите полосу пропускания LoRa на 125 КГц
    bandwidth = 125000

    # Установите мощность TX на 0 dBm (0.12 mW)
    txpower = 0

    # Выберите коэффициент расширения 7.
    Допустимый # диапазон от 5 до 12, где 5 #
    является самым быстрым, а 12 имеет # са-
    мый длинный диапазон.
    spreadingfactor = 7

    # Выберите скорость кодирования 5. Допустимый
    диапазон # от 5 до 8, где 5 # является самым
    быстрым, а 8 — самым длинным.
    codingrate = 5

    # Можно ограничить использование эфирного
    времени RNode с помощью следующих двух
    параметров конфигурации.
    # Краткосрочное ограничение применяется в
    # окне приблизительно 15 секунд,
    # а долгосрочное ограничение действует
    # в течение скользящего 60-минутного окна. Оба
    # параметра указываются в процентах.
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    # airtime_limit_long = 100
    # airtime_limit_short = 100
```

6.10 Последовательный интерфейс

Reticulum может использоваться напрямую через последовательные порты или через любое устройство с последовательным портом, которое будет прозрачно передавать данные.

Полезно для прямой связи по паре проводов или для использования таких устройств, как радиостанции для передачи данных и лазеры.

```
[[Serial Interface]]
  type = SerialInterface
  enabled = yes

  # Последовательный порт для устройства
  port = /dev/ttyUSB0

  # Установите скорость последовательной передачи данных и другие
  # параметры конфигурации.
  speed = 115200
  databits = 8
  parity = none
  stopbits = 1
```

6.11 Интерфейс канала

Используя этот интерфейс, Reticulum может применять любую программу в качестве интерфейса через *stdin* и *stdout*. Это можно использовать для простого создания виртуальных интерфейсов или для взаимодействия с пользовательским оборудованием или другими системами.

```
[[Pipe Interface]]
  type = PipeInterface
  enabled = yes

  # Внешняя команда для выполнения
  command = netcat -l 5757

  # Дополнительная задержка перезапуска, в секундах
  respawn_delay = 5
```

Reticulum будет записывать все пакеты в *stdin* опции `command` и непрерывно читать и сканировать её *stdout* на наличие пакетов Reticulum. Если достигается `EOF`, Reticulum попытается перезапустить программу после ожидания `respawn_interval` секунд.

6.12 Интерфейс KISS

С помощью интерфейса KISS вы можете использовать Reticulum через различные пакетные радиомодемы и TNC, включая OpenModem. Интерфейсы KISS также можно настроить на периодическую отправку маяков для идентификации станции.

Предупреждение

Радиочастотный спектр является законодательно контролируемым ресурсом, и законодательство в разных странах мира значительно различается. Вы несете ответственность за ознакомление с любыми соответствующими нормативными актами для вашего местоположения и за принятие решений соответствующим образом.

```
[[Пакетный радиоинтерфейс KISS]]
  type = KISSInterface
  enabled = yes

  # Последовательный порт для устройства
  port = /dev/ttyUSB1

  # Установите скорость последовательной передачи данных и другие
  # параметры конфигурации.
  speed = 115200
  databits = 8
  parity = none
  stopbits = 1

  # Установить преамбулу модема.
  preamble = 150

  # Установить TX-хвост модема.
  txtail = 10

  # Настроить параметры CDMA. Эти
  # настройки являются разумными значениями по умолчанию.
  persistence = 200
  slottime = 20

  # Вы можете настроить интерфейс для отправки
  # идентификации по каналу с
  # установленным интервалом, настроив
  # следующих двух параметров. Интерфейс KISS #
  будет идентифицировать себя только в том слу-
  чае, если с момента его последней # фактической
  передачи истёк заданный # интервал. Интервал
  # настраивается в секундах.
  # Эта опция закомментирована и по
  умолчанию не # используется.
  # id_callsign = MYCALL-0
  # id_interval = 600

  # Использовать ли управление потоком KISS.
  # Это полезно для модемов, которые имеют
  # небольшой внутренний буфер пакетов, но
  # вместо этого поддерживают управление потоком пакетов.
  flow_control = false
```

6.13 Интерфейс AX.25 KISS

Если вы используете Reticulum в любительском радиодиапазоне, вы можете воспользоваться интерфейсом AX.25 KISS. Таким образом, Reticulum будет автоматически инкапсулировать свой трафик в AX.25, а также идентифицировать передачи вашей станции с помощью вашего позывного и SSID.

Делайте это только в случае крайней необходимости! Reticulum не нуждается в уровне AX.25, и его использование влечет за собой дополнительные накладные расходы на каждый пакет для инкапсуляции в AX.25.

Более эффективный способ — использовать обычный KISS-интерфейс с описанной выше функцией маяка.

Предупреждение

Радиочастотный спектр является законодательно контролируемым ресурсом, и законодательство в разных странах мира значительно различается. Вы несете ответственность за ознакомление с любыми соответствующими нормативными актами для вашего местоположения и за принятие решений соответствующим образом.

```
[[Packet Radio AX.25 KISS Interface]]
  type = AX25KISSInterface

  # Установите позывной станции и SSID
  callsign = NO1CLL
  ssid = 0

  # Включите интерфейс, если хотите его использовать!
  enabled = yes

  # Последовательный порт для устройства
  port = /dev/ttyUSB2

  # Установите скорость последовательной передачи данных и другие
  # параметры конфигурации.
  speed = 115200
  databits = 8
  parity = none
  stopbits = 1

  # Установите преамбулу модема. Преамбула в 15
  0 мс # должна быть разумным # значением по
  умолчанию, но может потребоваться # её увели-
  чить для радиостанций с медленным # откры-
  тием шумоподавителя и длительным временем #
   переключения TX/RX.
  preamble = 150

  # Устанавливает хвостовую часть модема TX. В боль-
  шинстве # случаев ее следует держать как можно ниже
  #, чтобы не тратить эфирное время.
  txtail = 10

  # Настроить параметры CDMA. Эти
  # настройки являются разумными значениями по умолчанию.
  persistence = 200
  slottime = 20
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# Использовать ли управление потоком KISS.
# Это полезно для модемов с # небольшим внутренним
буфером пакетов.
flow_control = false
```

6.14 Общие параметры интерфейса

Ряд общих параметров конфигурации доступен на большинстве интерфейсов. Их можно использовать для управления различными аспектами поведения интерфейса.

- Параметр `enabled` сообщает Reticulum, следует ли поднимать интерфейс. По умолчанию установлено `False`. Для поднятия любого интерфейса параметр `enabled` должен быть установлен в `True` или `Yes`.
- Параметр `mode`позволяет выбрать высокоуровневое поведение интерфейса из нескольких вариантов.
  - Значение по умолчанию — `full`. В этом режиме доступны все функции обнаружения, маршрутизации и транспорта.
  - В режиме `access_point` (или сокращенно `ap`) интерфейс будет работать как точка доступа к сети. В этом режиме объявления не будут автоматически транслироваться на интерфейсе, и пути к назначениям на интерфейсе будут иметь гораздо более короткое время истечения срока действия. Этот режим полезен для создания интерфейсов, которые в основном неактивны, за исключением случаев, когда кто-то их фактически использует. Примером может служить радиоинтерфейс, обслуживающий широкую область, где пользователи, как ожидается, будут подключаться на короткое время, использовать сеть, а затем снова отключаться.
- Опция `outgoing` устанавливает, разрешена ли интерфейсу передача данных. По умолчанию установлено значение `True`. Если установлено значение `False` или `No`, интерфейс будет только принимать данные и никогда их не передавать.
- Опция `network_name` устанавливает имя виртуальной сети для интерфейса. Это позволяет нескольким отдельным сегментам сети существовать на одном и том же физическом канале или носителе.
- Опция `passphrase` устанавливает парольную фразу аутентификации на интерфейсе. Эту опцию можно использовать в сочетании с опцией `network_name` или отдельно.
- Опция `ifac_size` позволяет настраивать длину кодов аутентификации интерфейса, передаваемых каждым пакетом в именованных и/или аутентифицированных сетевых сегментах. По умолчанию она устанавливается на размер, подходящий для данного интерфейса, но с помощью этой опции ее можно установить на пользовательский размер от 8 до 512 бит. При обычном использовании этот параметр не должен отличаться от значения по умолчанию.
- Опция `announce_cap` позволяет настроить максимальную пропускную способность, выделяемую в любой момент времени для распространения объявлений и другого служебного трафика сети. По умолчанию он настроен на 2% и обычно не требует изменения. Может быть установлено любое значение в диапазоне от `1` до `100`.

  Если интерфейс превышает свой лимит объявлений, он будет ставить объявления в очередь для последующей передачи. Reticulum всегда будет отдавать приоритет распространению объявлений от ближайших узлов. Это гарантирует приоритет локальной топологии и предотвращает перегрузку медленных сетей взаимосвязанными быстрыми сетями.

  Направления, которые быстро переобъявляют себя, будут дополнительно деприоритизированы. Попытка получить «первое место» путём спама объявлениями будет иметь прямо противоположный эффект: перемещение в конец очереди каждый раз, когда от чрезмерно объявляющего направления поступает новое объявление.

  Это означает, что всегда выгодно выбирать сбалансированную скорость объявления и не объявлять чаще, чем это на самом деле необходимо для функционирования вашего приложения.
- Опция `bitrate` настраивает битрейт интерфейса. Reticulum будет использовать скорости интерфейса, сообщаемые оборудованием, или попытается определить подходящую скорость, если оборудование не сообщает никаких данных. В большинстве случаев автоматически найденной скорости должно быть достаточно, но её можно настроить с помощью опции `bitrate`, чтобы установить скорость интерфейса в *bits per second*.

6.15 Режимы интерфейса

Необязательный параметр `mode` доступен на всех интерфейсах и позволяет выбрать высокоуровневое поведение интерфейса из нескольких режимов. Эти режимы влияют на то, как Reticulum выбирает пути в сети, как распространяются объявления, как долго пути остаются действительными и как обнаруживаются пути.

Настройка режимов на интерфейсах не является строго обязательной, но может быть полезной при построении или подключении к более сложным сетям. Если ваш экземпляр Reticulum не является транспортным узлом, настройка режимов интерфейса редко бывает полезной, и в таких случаях интерфейсы обычно следует оставлять в режиме по умолчанию.

- Режим по умолчанию –полный. В этом режиме активируются все функции обнаружения, построения сетки и транспорта.
- Режим шлюза (или сокращенно `gw`) также имеет все доступные функции обнаружения, построения сетки и транспорта, но дополнительно будет пытаться обнаруживать неизвестные пути от имени других узлов, находящихся на интерфейсе шлюза . Если Reticulum получает запрос пути для неизвестного назначения от узла на интерфейсе шлюза , он попытается обнаружить этот путь через все другие активные интерфейсы и перешлет обнаруженный путь запрашивающей стороне, если таковой будет найден.

  Если вы хотите разрешить другим узлам широко разрешать пути или подключаться к сети через интерфейс, может быть полезно перевести его в этот режим. Создав цепочку интерфейсов шлюза , другие узлы смогут немедленно обнаруживать пути к любому назначению вдоль цепочки.

  Обратите внимание! Именно интерфейс, *обращенный к клиентам* , должен быть переведен в режим шлюза для работы этой функции, а не интерфейс, обращенный к более широкой сети (хотя для этого может быть полезен режим границы).

- В режиме `access_point` (или сокращенно `ap`) интерфейс будет работать как точка доступа к сети. В этом режиме объявления не будут автоматически транслироваться на интерфейсе, и пути к назначениям на интерфейсе будут иметь гораздо более короткое время истечения срока действия. Кроме того, запросы пути от клиентов на интерфейсе точки доступа будут обрабатываться так же, как и на интерфейсе шлюза .

  Этот режим полезен для создания интерфейсов, которые остаются неактивными, пока кто-то фактически не начнет их использовать. Примером может служить радиоинтерфейс, обслуживающий широкую область, где пользователи, как ожидается, будут подключаться на короткое время, использовать сеть, а затем снова отключаться.

- Режим роуминга следует использовать на интерфейсах, которые находятся в роуминге (физически мобильны), с точки зрения других узлов в сети. Например, если транспортное средство оснащено внешним интерфейсом LoRa и внутренним , основанным на WiFi интерфейсом, который обслуживает устройства, перемещающиеся вместе с транспортным средством, внешний интерфейс LoRa должен быть настроен как роуминг , а внутренний интерфейс может быть оставлен в режиме по умолчанию. При включенном транспорте такая настройка позволит всем внутренним устройствам связываться друг с другом, а также со всеми другими устройствами, доступными на LoRa-стороне сети, когда они находятся в пределах досягаемости. Устройства на LoRa-стороне сети также смогут связываться с устройствами внутри транспортного средства, когда оно находится в пределах досягаемости. Пути через роуминговые интерфейсы также истекают быстрее.
- Цель режима `boundary` — указать интерфейсы, которые устанавливают связь с сегментами сети, значительно отличающимися от того, на котором существует этот узел. Например, если экземпляр Reticulum является частью сети на основе LoRa, но также имеет высокоскоростное соединение с общедоступным транспортным узлом, доступным в интернете, интерфейс, подключающийся через интернет, должен быть установлен в режим `boundary` .

Таблицу, описывающую влияние всех режимов на распространение объявлений, см. в разделе *Правила распространения объявлений*.

6.16 Управление скоростью объявлений

Встроенные механизмы управления объявлениями и описанная выше опция `announce_cap` по умолчанию достаточны в большинстве случаев, но в некоторых случаях, особенно на быстрых интерфейсах, может быть полезно контролировать целевую скорость объявлений. Используя опции `announce_rate_target` , `announce_rate_grace` и `announce_rate_penalty` , это можно сделать для каждого интерфейса и регулирует *скорость*, *с которой полученные объявления повторно передаются на другие интерфейсы* .

- Опция `announce_rate_target` устанавливает минимальное количество времени (в секундах), которое должно пройти между полученными объявлениями для любого одного назначения. Например, установка этого значения на `3600` означает, что объявления, полученные на этом интерфейсе, будут повторно передаваться и распространяться на другие интерфейсы только раз в час, независимо от частоты их получения.
- Необязательный параметр `announce_rate_grace` определяет, сколько раз назначение может нарушить скорость объявлений до того, как будет применена целевая скорость.
- Необязательный параметр `announce_rate_penalty` конфигурирует дополнительное количество времени, которое добавляется к обычной целевой скорости. Например, если определён штраф в `7200` секунд, то после применения целевой скорости объявления от данного назначения будут распространяться только каждые 3 часа, пока оно не снизит свою фактическую скорость объявлений до целевой.

Эти механизмы, в сочетании с упомянутыми выше механизмами `annouce_cap`, означают, что крайне важно выбрать сбалансированную стратегию объявлений для ваших назначений. Чем более сбалансированным вы сможете принять это решение , тем легче вашим адресатам будет попасть в более медленные сети, находящиеся на расстоянии многих переходов. Или вы можете расставить приоритеты, достигая только высокопроизводительных сетей с более частыми объявлениями.

Текущую статистику и информацию о скорости объявлений можно просмотреть с помощью команды`rnpath -r` .

Важно отметить, что не существует единственно правильного или неправильного способа настройки скорости объявлений. Более медленные сети, естественно, будут стремиться использовать менее частые объявления для экономии пропускной способности, в то время как очень быстрые сети могут поддерживать приложения, которым требуются очень частые объявления. Reticulum реализует эти механизмы для обеспечения бесшовного сосуществования и взаимосвязи большого количества типов сетей.

6.17 Ограничение скорости для новых адресатов

На публичных интерфейсах, где любой может подключаться и объявлять новые адресаты, может быть полезно контролировать скорость обработки объявлений для новых адресатов.

Если большой приток объявлений для вновь созданных или ранее неизвестных адресатов происходит в течение короткого промежутка времени, Reticulum приостановит обработку этих объявлений, чтобы трафик объявлений для известных и ранее установленных адресатов мог продолжать обрабатываться без перебоев.

После того, как всплеск спадет и пройдет дополнительный период ожидания, задержанные объявления будут публиковаться с медленной скоростью, пока очередь ожидания не будет очищена. Это также означает, что если узел решит подключиться к публичному интерфейсу, объявить большое количество поддельных адресов, а затем отключиться, эти адреса никогда не попадут в таблицы маршрутов и не будут расходовать пропускную способность сети на повторную передачу объявлений.

Важно отметить что контроль входящего трафика работает на уровне *отдельных под-интерфейсов* . Например, это означает, что один клиент на *TCP Server Interface* не может нарушить обработку входящих объявлений для других подключенных клиентов на том же *TCP Server Interface* . Все остальные клиенты на том же интерфейсе по-прежнему будут получать обработанные новые объявления без перерыва.

По умолчанию Reticulum будет обрабатывать это автоматически, и контроль входящих объявлений будет включен на интерфейсах, где это целесообразно. Обычно не должно быть необходимости изменять конфигурацию контроля входящего трафика, но все параметры доступны для настройки при необходимости.

- Опция `ingress_control` указывает Reticulum, следует ли включать контроль входящих объявлений на интерфейсе. По умолчанию установлено значение `True` .

- Опция `ic_new_time` конфигурирует, как долго (в секундах) интерфейс считается недавно созданным. По умолчанию установлено `2*60*60` секунд. Эта опция полезна для общедоступных интерфейсов, которые создают новые подынтерфейсы при подключении нового клиента.
- Опция`ic_burst_freq_new`устанавливает максимальную частоту входящих объявлений для недавно созданных интерфейсов. По умолчанию установлено `3.5`объявлений в секунду.
- Опция `ic_burst_freq` устанавливает максимальную частоту входящих объявлений для других интерфейсов. По умолчанию установлено `12` объявлений в секунду.

  Если интерфейс превышает свою пиковую частоту, входящие объявления для неизвестных адресатов будут временно удерживаться в очереди и не будут обрабатываться до более позднего времени.

- Опция `ic_max_held_announces` устанавливает максимальное количество уникальных объявлений, которые будут храниться в очереди. Любые дополнительные уникальные объявления будут отброшены. По умолчанию `256` объявлений.
- Опция `ic_burst_hold` устанавливает, сколько времени (в секундах) должно пройти после того, как частота всплеска опустится ниже порогового значения, чтобы всплеск объявлений считался очищенным. По умолчанию `60` секунд.
- Опция `ic_burst_penalty` устанавливает, сколько времени (в секундах) должно пройти после того, как всплеск считается очищенным, прежде чем удерживаемые объявления могут начать освобождаться из очереди. По умолчанию `5*60` секунд.
- Опция `ic_held_release_interval` устанавливает, сколько времени (в секундах) должно пройти между освобождением каждого удерживаемого объявления из очереди. По умолчанию `30` секунд.

ГЛАВА

СЕМЬ

СОЗДАНИЕ СЕТЕЙ

Эта глава предоставит вам знания, необходимые для построения сетей с Reticulum, что часто может быть проще, чем использование традиционных стеков, поскольку вам не нужно беспокоиться о согласовании адресов, подсетей и маршрутизации для всей сети, о развитии которой в будущем вы можете не знать. С Reticulum вы можете просто добавлять новые сегменты в вашу сеть по мере необходимости, и Reticulum автоматически обеспечит конвергенцию всей сети.

7.1 Концепции и обзор

При построении сетей с Reticulum необходимо учитывать следующие важные моменты:

- В сети Reticulum любой узел может автономно генерировать столько адресов (называемых *назначениями* в терминологии Reticulum), сколько ему необходимо; эти адреса становятся глобально достижимыми для остальной части сети. Централизованного контроля над адресным пространством не существует.
- Reticulum был разработан для работы как с очень малыми, так и с очень большими сетями. Хотя адресное пространство может поддерживать миллиарды конечных точек, Reticulum также очень полезен, когда требуется связь между всего несколькими устройствами.
- Низкоскоростные сети, такие как LoRa и пакетное радио, могут беспрепятственно взаимодействовать и соединяться с гораздо более крупными и высокоскоростными сетями. Reticulum автоматически управляет потоком информации к и от различных сетевых сегментов, а при ограниченной пропускной способности приоритет отдается локальному трафику.
- Reticulum по умолчанию обеспечивает анонимность отправителя/инициатора. Отсутствует возможность фильтровать трафик или дискриминировать его по источнику.
- Весь трафик шифруется с использованием эфемерных ключей, генерируемых обменом ключами Диффи-Хеллмана на эллиптической кривой Curve25519. Невозможно проверять содержимое трафика, приоритизировать или регулировать его определенные виды. Все транспортные и маршрутизирующие уровни, таким образом, полностью независимы от типа трафика и будут передавать весь трафик одинаково.
- Reticulum может функционировать как с инфраструктурой, так и без нее. Когда доступны транспортные узлы, они могут маршрутизировать трафик для других узлов через несколько прыжков и будут функционировать как распределенное криптографическое хранилище ключей. Если транспортные узлы недоступны, все узлы, находящиеся в пределах дальности связи, по-прежнему могут обмениваться данными.
- Каждый узел может стать транспортным узлом, просто включив его в своей конфигурации, но нет необходимости, чтобы каждый узел в сети был транспортным узлом. Предоставление каждому узлу роли транспортного узла в большинстве случаев ухудшит производительность и надежность сети.

    > *В общих чертах, если узел стационарен, хорошо подключен и работает большую часть времени, он является хорошим кандидатом на роль транспортного узла. Для оптимальной производительности сеть должна содержать количество транспортных узлов, достаточное для обеспечения связности с предполагаемой областью/топографией, и ненамного больше.*

- Reticulum разработан для надежной работы в открытых, недоверенных средах. Это означает, что вы можете использовать его для создания сетей с открытым доступом, где участники могут свободно и неорганизованно присоединяться и покидать их. Это свойство

позволяет создать совершенно новый и до сих пор в основном неисследованный класс сетевых приложений , где сети и информационные потоки внутри них могут формироваться и распадаться органически.

- Вы также легко можете создавать закрытые сети, поскольку Reticulum позволяет добавлять аутентификацию к любому интерфейсу. Это означает, что вы можете ограничить доступ к любому типу интерфейса, даже при использовании устаревших устройств, таких как модемы. Вы также можете смешивать аутентифицированные и открытые интерфейсы в одной системе. См. раздел «Параметры общего интерфейса» главы «Интерфейсы» данного руководства для получения информации о том, как настроить аутентификацию интерфейса.

Reticulum позволяет объединять различные типы сетевых сред в единую mesh-сеть или использовать одну среду . Вы можете построить «виртуальную сеть», полностью работающую через Интернет, где все узлы взаимодействуют по TCP и UDP «каналам». Вы также можете построить такую сеть, используя другие уже установленные каналы связи в качестве базового носителя для Reticulum.

Однако большинство реальных сетей, вероятно, будут использовать ту или иную форму беспроводной или прямой проводной связи. Чтобы Reticulum мог обмениваться данными по любому типу среды, вы должны указать его в файле конфигурации, по умолчанию расположенном по адресу `~/.reticulum/config` . См. главу «Поддерживаемые интерфейсы» данного руководства для примеров конфигурации интерфейса.

Может быть настроено любое количество интерфейсов, и Reticulum автоматически решит, какие из них подходят для использования в любой конкретной ситуации, в зависимости от того, куда должен поступать трафик.

7.2 Примеры сценариев

Этот раздел иллюстрирует несколько примеров сценариев и описывает общие принципы их планирования, реализации и конфигурирования.

7.2.1 Взаимосвязанные LoRa-узлы

Организация стремится предоставлять услуги связи и информации своим членам, расположенным преимущественно в трёх отдельных областях. Были найдены три подходящие места на вершинах холмов для установки оборудования организацией: Узел A, B и C.

Поскольку основной объём данных, которыми обмениваются пользователи, является текстовым, требования к пропускной способности низкие, и для подключения пользователей к сети выбраны радиостанции LoRa.

Благодаря найденным расположениям на вершинах холмов обеспечивается прямая радиовидимость между узлами A и B, а также между узлами B и C. Вследствие этого организации не требуется использовать Интернет для соединения узлов; вместо этого она приобретает четыре радиостанции Point-to-Point на основе WiFi для их взаимосвязи.

На каждом узле установлен Raspberry Pi, функционирующий как шлюз. Радио LoRa подключено к Raspberry Pi с помощью USB-кабеля, а радио WiFi подключено к порту Ethernet Raspberry Pi. На участке B для связи как с участком A, так и с участком C требуются два радиомодуля WiFi, поэтому к Raspberry Pi в этом месте подключается дополнительный адаптер Ethernet.

После установки оборудования Reticulum инсталлируется на все Raspberry Pi, а на участках A и C добавляется по одному интерфейсу для радио LoRa и для радио WiFi. На участке B в файл конфигурации Reticulum добавляется один интерфейс для радио LoRa и по одному интерфейсу для каждого радио WiFi. Опция транспортного узла включена в конфигурациях всех трех шлюзов.

Сеть теперь функционирует и готова обслуживать пользователей во всех трех областях. Организация подготавливает радио LoRa, которое поставляется конечным пользователям вместе с файлом конфигурации Reticulum, содержащим правильные параметры для связи с радиостанциями LoRa, установленными на узлах шлюзов.

Как только пользователи подключатся к сети, каждый сможет общаться с любым другим на всех трех площадках.

7.2.2 Мостовое соединение через Интернет

По мере роста организации формируется несколько новых сообществ в местах, слишком удаленных от основной сети, чтобы быть доступными по каналам Wi-Fi. Новые шлюзы, аналогичные ранее установленным, настраиваются для новых сообществ на новых площадках D и E, но они изолированы от основной сети и обслуживают только локальных пользователей.

После изучения вариантов выясняется, что возможно установить интернет-соединение на площадке A, и на Raspberry Pi на площадке A настраивается интерфейс для Reticulum через это интернет-соединение.

Член организации на площадке D, по имени Дори, готова помочь, поделившись уже имеющимся у нее дома интернет-соединением , и может оставить Raspberry Pi работающим. На ее Raspberry Pi настраивается новый интерфейс Reticulum, подключающийся к недавно включенному интернет-интерфейсу на шлюзе на площадке A. Дори теперь подключена как к узлам на ее собственном локальном сайте (через LoRa-шлюз на вершине холма), так и ко всем объединенным пользователям площадок A, B и C. Затем она включает транспорт на своем узле, и трафик с площадки D теперь может достигать всех на площадках A, B и C, и наоборот.

7.2.3 Рост и конвергенция

По мере роста организации добавляется больше шлюзов, чтобы не отставать от растущей пользовательской базы. Некоторые локальные шлюзы даже добавляют УКВ-радиостанции и пакетные модемы для охвата отдалённых пользователей и сообществ, находящихся вне зоны действия радиостанций LoRa и каналов WiFi.

По мере подключения большего количества сайтов, шлюзов и пользователей требуемое количество координации сводится к минимуму. Если одно сообщество хочет добавить возможность подключения к соседнему, это можно сделать просто, не вовлекая всех и не координируя адресное пространство или таблицы маршрутизации.

С расширением географического охвата операторы на сайте A однажды обнаруживают, что исходные интерфейсы, подключённые к Интернету, больше не используются. Сеть конвергировала, став полностью самосвязанной, и сайты, которые когда-то были плохо связанными периферийными узлами, теперь являются неотъемлемой частью сети.

ГЛАВА

ВОСЕМЬ

ПОДДЕРЖКА RETICULUM

Вы можете помочь поддержать постоянное развитие открытых, свободных и частных систем связи , делая пожертвования, предоставляя обратную связь и внося вклад в код и учебные ресурсы.

8.1 Пожертвования

Пожертвования с благодарностью принимаются по следующим каналам:

```
Monero:
84FpY1QbxHcgdseePYNmhTHcrgMX4nFfBYtz2GKYToqHVVhJp8Eaw1Z1EedRnKD19b3B8NiLCGVxzKV17UMmmeEsCrPyA5w

Ethereum:
0x81F7B979fEa6134bA9FD5c701b3501A2e61E897a

Bitcoin:
3CPmacGm34qYvR6XWLVEJmi2aNe3PZqUuq

Liberapay:
https://liberapay.com/Reticulum/

Ko-Fi:
https://ko-fi.com/markqvist
```

Являются ли определенные функции в дорожной карте разработки важными для вас или вашей организации? Воплотите их в жизнь быстрее, спонсируя их реализацию.

8.2 Предоставить обратную связь

Все отзывы об использовании, функционировании и потенциальных сбоях любых компонентов системы очень ценны для дальнейшего развития и улучшения Reticulum.

Reticulum ни при каких обстоятельствах не собирает и не передаёт никаких автоматизированных аналитических данных, телеметрии, отчётов об ошибках или статистики, поэтому мы полагаемся на традиционную обратную связь от людей.

8.3 Внесение кода

Присоединяйтесь к нам в репозитории GitHub, чтобы сообщать о проблемах, предлагать функциональность и вносить свой вклад в разработку Reticulum.

ГЛАВА

ДЕВЯТЬ

ПРИМЕРЫ КОДА

Ряд примеров включён в исходный дистрибутив Reticulum. Вы можете использовать их для изучения принципов написания собственных программ.

9.1 Минимальный

Пример *Minimal* демонстрирует минимально необходимую настройку для подключения к сети Reticulum из вашей программы. Примерно в пяти строках кода вы инициализируете Сетевой стек Reticulum, и он будет готов к передаче трафика в вашей программе.

```
##########################################################
# Этот пример RNS демонстрирует минимальную настройку, которая  #
 # запустит Сетевой стек Reticulum, сгенерирует #
# новое назначение и позволит пользователю отправить объявление.  #
##########################################################

import argparse
import sys
import RNS

# Давайте определим имя приложения. Мы будем использовать
это для всех # назначений, которые мы создаем. Поскольку
этот базовый пример # является частью ряда примеров утилит
, мы поместим # их все в пространство имен приложения "
example_utilities" APP_NAME = "example_utilities"

# Эта инициализация выполняется при запуске программы
def program_setup(configpath):
    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Случайно создадим новую личность для нашего примера
    identity = RNS.Identity()

    # Используя только что созданную личность, мы создаем пункт назначения.
    # Назначения — это конечные точки в Reticulum, которые могут
    быть адресованы # и с которыми можно обмениваться данными.
    Пункты назначения также могут объявлять о своем # существовании,
     что позволит сети узнать, что они доступны, # и автоматически
    создавать пути к ним из любого другого места # в сети.
    destination = RNS.Destination(
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        идентификатор,
        RNS.Destination.IN,
        RNS.Destination.SINGLE,
        APP_NAME,
        "minimalsample"
    )

    # Мы настраиваем место назначения для автоматического подтверждения всех # пакетов, адресованных ему. Таким образом, RNS будет автоматически # генерировать подтверждение для каждого входящего пакета и передавать его # обратно отправителю этого пакета. Это позволит любому, кто # пытается связаться с местом назначения, узнать, было ли его # сообщение получено правильно.
    destination.set_proof_strategy(RNS.Destination.PROVE_ALL)

    # Всё готово!
    # Давайте передадим управление циклу анонсирования
    announceLoop(destination)


def announceLoop(destination):
    # Дайте пользователю знать, что всё готово
    RNS.log(
        "Минимальный пример "+
        RNS.prettyhexrep(destination.hash)+
        " работает, нажмите Enter, чтобы вручную отправить объявление (Ctrl-C для выхода)"
    )

    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажимает Enter, мы объявим наше сервер# назначение в сети, что позволит клиентам # узнать, как создавать сообщения, направленные на него.
    while True:
        entered = input()
        destination.announce()
        RNS.log("Отправлено объявление от "+RNS.prettyhexrep(destination.hash))


# ##########################################################
# #### Запуск программы #####################################
# ##########################################################

# Эта часть программы запускается при старте,
# разбирает ввод пользователя, а затем запускает # желаемый режим программы.
if __name__ == "__main__":
    try:
        parser = argparse.ArgumentParser(
            description="Минимальный пример для запуска Reticulum и создания назначения"
        )

        parser.add_argument(
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            "--config",
            action="store",
            default=None,
            help="путь к альтернативному каталогу конфигурации Reticulum",
            type=str
        )

        args = parser.parse_args()

        if args.config:
            configarg = args.config
        else:
            configarg = None

        program_setup(configarg)

    except KeyboardInterrupt:
        print("")
        sys.exit(0)
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/Minimal.py.

9.2 Объявление

Пример с объявлениями основывается на предыдущем примере, исследуя, как объявить назначение в сети и как позволить вашей программе получать уведомления об объявлениях от соответствующих назначений.

```
##########################################################
# Этот пример RNS демонстрирует настройку объявления      # # об-
ратных вызовов, которые позволят приложению получать  # # уве-
домления, когда приходит соответствующее объявление # ########
##########################################################

import argparse
import random
import sys
import RNS

# Давайте определим имя приложения. Мы будем использовать
это для всех # назначений, которые мы создаем. Поскольку
этот базовый пример # является частью ряда примеров утилит
, мы поместим # их все в пространство имен приложения "
example_utilities" APP_NAME = "example_utilities"

# Мы инициализируем два списка строк для использования в качестве app_data
fruits = ["Peach", "Quince", "Date", "Tangerine", "Pomelo", "Carambola", "Grape"]
noble_gases = ["Helium", "Neon", "Argon", "Krypton", "Xenon", "Radon", "Oganesson"]

# Эта инициализация выполняется при запуске программы
def program_setup(configpath):
    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    # Случайно создадим новую личность для нашего примера
    identity = RNS.Identity()

    # Используя только что созданную нами идентификацию, мы создаем две точки назначения
    # в пространстве приложения "example_utilities.announcesample".
    #
    # Назначения — это конечные точки в Reticulum, которые могут
    быть адресованы # и с которыми можно обмениваться данными.
    Пункты назначения также могут объявлять о своем # существовании,
     что позволит сети узнать, что они доступны, # и автоматически
    создавать пути к ним из любого другого места # в сети.
    destination_1 = RNS.Destination(
        identity, RNS.
        Destination.IN, RNS.
        Destination.SINGLE, APP_
        NAME,
        “announcesample”,
        “fruits”
    )

    destination_2 = RNS.Destination(
        identity, RNS.
        Destination.IN, RNS.
        Destination.SINGLE, APP_
        NAME,
        “announcesample”,
        “noble_gases”
    )

    # Мы настраиваем точки назначения на автоматическое подтвер-
    ждение всех # пакетов, адресованных им. Таким образом, RNS авто-
    матически # сгенерирует подтверждение для каждого входящего па-
    кета и передаст его # обратно отправителю этого пакета. Это поз-
    волит любому, кто # пытается связаться с местом назначения,
    узнать, было ли его # сообщение получено правильно.
    destination_1.set_proof_strategy(RNS.Destination.PROVE_ALL)
    destination_2.set_proof_strategy(RNS.Destination.PROVE_ALL)

    # Мы создаем обработчик объявлений и настраиваем его на запрос объ-
    явлений только от "example_utilities.announcesample.fruits". # По-
    пробуйте изменить фильтр и посмотрите, что произойдет.
    announce_handler = ExampleAnnounceHandler(
        aspect_filter="example_utilities.announcesample.fruits"
    )

    # Мы регистрируем обработчик объявлений в Reticulum RNS.
    Transport.register_announce_handler(announce_handler)

    # Всё готово!
    # Давайте передадим управление циклу анонсирования
    announceLoop(destination_1, destination_2)
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
def announceLoop(destination_1, destination_2):
    # Дайте пользователю знать, что всё готово
    RNS.log("Пример объявления запущен, нажмите Enter, чтобы отправить объявление вручную (Ctrl-C, чтобы␣
↪выйти)")

    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажимает Enter, мы объявим наше сер-
    верное # назначение в сети, что позволит клиентам # узнать,
     как создавать сообщения, направленные на него.
    while True:
        entered = input()

        # Случайно выбираем фрукт
        fruit = fruits[random.randint(0,len(fruits)-1)]

        # Отправляем объявление, включая данные приложения
        destination_1.announce(app_data=fruit.encode("utf-8"))
        RNS.log(
            "Отправлено объявление от "+RNS.
            prettyhexrep(destination_1.hash)+" ("+
            destination_1.name+")"
        )

        # Случайно выбираем благородный газ
        noble_gas = noble_gases[random.randint(0,len(noble_gases)-1)]

        # Отправляем объявление, включая данные приложения
        destination_2.announce(app_data=noble_gas.encode("utf-8"))
        RNS.log(
            "Отправлено объявление от "+RNS.
            prettyhexrep(destination_2.hash)+" ("+
            destination_2.name+")"
        )

# Нам потребуется определить класс обработчика объявлений, который
# Reticulum может использовать для отправки сообщений при получении объявления.
class ExampleAnnounceHandler:
    # Метод инициализации принимает необязательный аргу-
    мент # aspect_filter. Если aspect_filter установлен в # None
    , все объявления будут переданы экземпляру.
    # Если требуются только определённые объявления, его можно установить в
    # строковый аспект.
    def __init__(self, aspect_filter=None):
        self.aspect_filter = aspect_filter

    # Этот метод будет вызван транспортной системой Reticulum #
    при поступлении объявления, соответствующего # настроенному
    фильтру аспектов. Фильтры должны быть специфичными # и не
    могут использовать подстановочные знаки.
    def received_announce(self, destination_hash, announced_identity, app_data):
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        RNS.log(
            "Получено объявление от "+RNS.
            prettyhexrep(destination_hash)
        )

        if app_data:
            RNS.log(
                "The announce contained the following app data: "+
                app_data.decode("utf-8")
            )

############################################################
#### Запуск программы ######################################
############################################################

# Эта часть программы запускается при старте,
# разбирает ввод пользователя, а затем запускает #
 желаемый режим программы.
if __name__ == "__main__":
    try:
        parser = argparse.ArgumentParser(
            description="Reticulum example that demonstrates announces and announce␣
↪handlers"
        )

        parser.add_argument(
            "--config",
            action="store",
            default=None,
            help="путь к альтернативному каталогу конфигурации Reticulum",
            type=str
        )

        args = parser.parse_args()

        if args.config:
            configarg = args.config
        else:
            configarg = None

        program_setup(configarg)

    except KeyboardInterrupt:
        print("")
        sys.exit(0)
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/Announce.py.

9.3 Трансляция

Пример *Broadcast*исследует, как передавать широковещательные сообщения с открытым текстом по сети.

```
############################################################
# Этот пример RNS демонстрирует трансляцию незашифрованной #
# информации для любых слушающих адресатов.                #
############################################################

import sys
import argparse
import RNS

# Давайте определим имя приложения. Мы будем использовать
это для всех # назначений, которые мы создаем. Поскольку
этот базовый пример # является частью ряда примеров утилит
, мы поместим # их все в пространство имен приложения "
example_utilities" APP_NAME = "example_utilities"

# Эта инициализация выполняется при запуске программы
def program_setup(configpath, channel=None):
    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Если пользователь не выбрал "канал", мы используем # канал
    по умолчанию, называемый "public_information".
    # Этот "канал" добавляется в пространство имен на-
    значения, # поэтому пользователь может выбирать
    различные широковещательные # каналы.
    if channel == None:
        channel = "public_information"

    # Мы создаем PLAIN-получателя. Это незашифрованная конечная точка, # ко-
    торую любой может прослушивать и отправлять ей информацию.
    broadcast_destination = RNS.Destination(
        None,
        RNS.Destination.IN,
        RNS.Destination.PLAIN,
        APP_NAME,
        "broadcast",
        channel
    )

    # Мы указываем функцию обратного вызова, которая будет вызываться каждый раз,
    # когда получатель принимает данные.
    broadcast_destination.set_packet_callback(packet_callback)

    # Всё готово!
    # Давайте передадим управление основному циклу
    broadcastLoop(broadcast_destination)

def packet_callback(data, packet):
    # Просто выводим полученные данные
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    print("")
    print("Полученные данные: "+data.decode("utf-8")+" \r\n > ",
    end="") sys.stdout.flush()

def broadcastLoop(destination):
    # Сообщаем пользователю, что все готово
    RNS.log(
        "Пример широковещания "+
        RNS.prettyhexrep(destination.hash)+
        " запущен, введите текст и нажмите Enter для отправки (Ctrl-C для выхода)"
    )

    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажимает Enter, мы отправляем ин-
    формацию, # которую он ввел в приглашении.
    while True:
        print("> ", end="")
        entered = input()

        if entered != "":
            data    = entered.encode("utf-8")
            packet  = RNS.Packet(destination, data)
            packet.send()



############################################################
#### Запуск программы ######################################
############################################################

# Эта часть программы запускается при старте,
# анализирует ввод пользователя, а затем запускает
# программу.
if __name__ == "__main__":
    try:
        parser = argparse.ArgumentParser(
            description="Пример Reticulum, демонстрирующий отправку и получение широковещательных сообщений
↪"
        )

        parser.add_argument(
            "--config",
            action="store",
            default=None,
            help="путь к альтернативному каталогу конфигурации Reticulum",
            type=str
        )

        parser.add_argument(
            "--channel",
            action="store",
            default=None,
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            help="имя широковещательного канала",
            type=str
        )

        args = parser.parse_args()

        if args.config:
            configarg = args.config
        else:
            configarg = None

        if args.channel:
            channelarg = args.channel
        else:
            channelarg = None

        program_setup(configarg, channelarg)

    except KeyboardInterrupt:
        print("")
        sys.exit(0)
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/Broadcast.py.

9.4 Эхо

Пример *Echo* демонстрирует связь между двумя адресатами с использованием интерфейса Packet.

```
##########################################################
# Этот пример RNS демонстрирует простую клиент/серверную
 # # эхо-утилиту. Клиент может отправить эхо-запрос на # #
 сервер, и сервер ответит, подтверждая получение # # пакета
.                                                          #
##########################################################

import argparse
import sys
import RNS

# Давайте определим имя приложения. Мы будем использовать
это для всех # назначений, которые мы создаем. Поскольку
этот пример эха # является частью набора примеров утилит,
мы 'll поместим # их все в пространство имен приложения "
example_utilities" APP_NAME = "example_utilities"


##########################################################
#### Серверная часть #####################################
##########################################################

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве сервера
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
def server(configpath):
    global reticulum

    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Случайно создаем новую идентичность для нашего эхо-сервера
    server_identity = RNS.Identity()

    # Мы создаем назначение, которое клиенты могут запрашивать. Мы хотим
    # иметь возможность проверять эхо-ответы нашим клиентам, поэтому мы
    # создаем «одно» назначение, которое может получать зашифрованные
    # сообщения. Таким образом, клиент может отправить запрос и быть
    # уверенным, что никто, кроме этого назначения, не смог
    # его прочитать.
    echo_destination = RNS.Destination(
        server_identity,
        RNS.Destination.IN,
        RNS.Destination.SINGLE,
        APP_NAME,
        "echo",
        "request"
    )

    # Мы настраиваем получателя на автоматическое подтверждение
    всех # пакетов, адресованных ему. Таким образом, RNS будет автома-
    тически # генерировать подтверждение для каждого входящего пакета
    и передавать его # обратно отправителю этого пакета.
    echo_destination.set_proof_strategy(RNS.Destination.PROVE_ALL)

    # Сообщите получателю, какую функцию в нашей программе # запус-
    кать при получении пакета. Мы делаем это, чтобы мы могли # выво-
    дить сообщение в журнал, когда сервер получает запрос.
    echo_destination.set_packet_callback(server_callback)

    # Всё готово!
    # Пусть's Ждать запросов клиента или ввода пользователя.
    announceLoop(echo_destination)


def announceLoop(destination):
    # Дайте пользователю знать, что всё готово
    RNS.log(
        "Сервер эхо-запросов "+
        RNS.prettyhexrep(destination.hash)+
        " работает, нажмите Enter, чтобы вручную отправить объявление (Ctrl-C для выхода)"
    )

    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажимает Enter, мы объявим наше сер-
    верное # назначение в сети, что позволит клиентам # узнать,
     как создавать сообщения, направленные на него.
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    while True:
        entered = input()
        destination.announce()
        RNS.log("Отправлено объявление от "+RNS.prettyhexrep(destination.hash))


def server_callback(message, packet):
    global reticulum

    # Сообщаем пользователю, что мы получили эхо-запрос, и # что
    мы собираемся отправить ответ отправителю запроса.
    # Отправка подтверждения обрабатывается автоматически, так как мы #
    настроили получателя на подтверждение всех входящих пакетов.

    reception_stats = ""
    if reticulum.is_connected_to_shared_instance:
        reception_rssi = reticulum.get_packet_rssi(packet.packet_hash)
        reception_snr  = reticulum.get_packet_snr(packet.packet_hash)

        if reception_rssi != None:
            reception_stats += " [RSSI "+str(reception_rssi)+" dBm]"

        if reception_snr != None:
            reception_stats += " [SNR "+str(reception_snr)+" dBm]"

    else:
        if packet.rssi != None:
            reception_stats += " [RSSI "+str(packet.rssi)+" dBm]"

        if packet.snr != None:
            reception_stats += " [SNR "+str(packet.snr)+" dB]"

    RNS.log("Получен пакет от эхо-клиента, подтверждение отправлено"+reception_stats)


############################################################
#### Часть клиента #########################################
############################################################

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве клиента
def client(destination_hexhash, configpath, timeout=None):
    global reticulum

    # Нам необходимо бинарное представление хеша назна-
    чения, # который был введен в командной строке
    try:
        dest_len = (RNS.Reticulum.TRUNCATED_HASHLENGTH//8)*2
        if len(destination_hexhash) != dest_len:
            raise ValueError(
                "Длина назначения недействительна, должно быть {hex} шестнадцатеричных символов (
↪{byte} байта).".format(hex=dest_len, byte=dest_len//2)
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            )

        destination_hash = bytes.fromhex(destination_hexhash)
    except Exception as e:
        RNS.log("Введено неверное назначение. Проверьте ваш ввод!") RNS.log(str(e)+
        " \n ") sys.exit(0)


    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Мы переопределяем уровень логирования для предо-
    ставления обратной связи при получении объявления.
    if RNS.loglevel < RNS.LOG_INFO:
        RNS.loglevel = RNS.LOG_INFO

    # Сообщаем пользователю, что клиент готов!
    RNS.log(
        "Эхо-клиент готов, нажмите Enter, чтобы отправить эхо-
        запрос на "+ destination_hexhash+ " (Ctrl-C
        для выхода)"
    )

    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажмет Enter, мы попытаемся отправить
    # эхо-запрос в пункт назначения, указанный в
    # командной строке.
    while True:
        input()

        # Давайте ' сначала проверим, знает ли RNS путь к месту назначения.
        # Если да, то мы ' загрузим идентификатор сервера и создадим пакет.
        if RNS.Transport.has_path(destination_hash):

            # Чтобы обратиться к серверу, нам нужно знать его ' открытый # ключ,
            поэтому мы проверяем, знает ли Reticulum это место назначения.
            # Это делается вызовом метода «recall» модуля #
            Identity. Если пункт назначения известен, он # вернет эк-
            земпляр Identity, который может быть использован
            в # исходящих пунктах назначения.
            server_identity = RNS.Identity.recall(destination_hash)

            # Мы получили правильный экземпляр идентификатора из метода
            # recall, поэтому давайте ' создадим исходящий # пункт назна-
            чения. Мы используем соглашение об именовании:
            # example_utilities.echo.request
            # Это соответствует именованию, которое мы указали в
            # серверной части кода.
            request_destination = RNS.Destination(
                server_identity,
                RNS.Destination.OUT,
                RNS.Destination.SINGLE,
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
                APP_NAME,
                "echo",
                "request"
            )

            # Адресат готов, так что давайте создадим пакет.
            # Мы устанавливаем адресат как request_destination,
            # который был только что создан, и единственные данные, которые мы добавляем,
            # это случайный хеш.
            echo_request = RNS.Packet(request_destination, RNS.Identity.get_random_
↪hash())

            # Отправьте пакет! Если пакет будет успешно # от-
            правлен, он вернет экземпляр PacketReceipt.
            packet_receipt = echo_request.send()

            # Если пользователь указал таймаут, мы устанавливаем этот
            # таймаут для квитанции пакета и настраиваем
            # функцию обратного вызова, которая будет вы-
            звана, если # пакет истечет.
            if timeout != None:
                packet_receipt.set_timeout(timeout)
                packet_receipt.set_timeout_callback(packet_timed_out)

            # Затем мы можем установить обратный вызов доставки для квитанции.
            # Это будет автоматически вызвано, когда от адресата будет
            получено подтверждение для # этого конкретного пакета.
            packet_receipt.set_delivery_callback(packet_delivered)

            # Сообщить пользователю, что эхо-запрос был отправлен
            RNS.log("Отправлен эхо-запрос на "+RNS.prettyhexrep(request_destination.hash)) else
        :
            # Если этот пункт назначения неизвестен, сообщите # поль-
            зователю, чтобы он дождался прибытия объявления.
            RNS.log("Пункт назначения пока неизвестен. Запрашиваю путь...") RNS.
            log("Нажмите Enter, чтобы повторить попытку вручную после получения
            объявления.") RNS.Transport.request_path(destination_hash)

# Эта функция вызывается, когда наш пункт назначения
для ответа # получает пакет подтверждения.
def packet_delivered(receipt):
    global reticulum

    if receipt.status == RNS.PacketReceipt.DELIVERED:
        rtt = receipt.get_rtt()
        if (rtt >= 1):
            rtt = round(rtt, 3)
            rttstring = str(rtt)+" секунд"
        else :
            rtt = round(rtt*1000, 3) rttstring
            = str(rtt)+" миллисекунд"
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        reception_stats = ""
        if reticulum.is_connected_to_shared_instance:
            reception_rssi = reticulum.get_packet_rssi(receipt.proof_packet.packet_hash)
            reception_snr  = reticulum.get_packet_snr(receipt.proof_packet.packet_hash)

            if reception_rssi != None:
                reception_stats += " [RSSI "+str(reception_rssi)+" dBm]"

            if reception_snr != None:
                reception_stats += " [SNR "+str(reception_snr)+" dB]"

        else:
            if receipt.proof_packet != None:
                if receipt.proof_packet.rssi != None:
                    reception_stats += " [RSSI "+str(receipt.proof_packet.rssi)+" dBm]"

                if receipt.proof_packet.snr != None:
                    reception_stats += " [SNR "+str(receipt.proof_packet.snr)+" dB]"

        RNS.log(
            "Получен действительный ответ от "+RNS.
            prettyhexrep(receipt.destination.hash)+",
            время кругового пути составляет "+
            rttstring+reception_stats
        )

# Эта функция вызывается, если истекает время ожидания пакета.
def packet_timed_out(receipt):
    if receipt.status == RNS.PacketReceipt.FAILED:
        RNS.log("Packet "+RNS.prettyhexrep(receipt.hash)+" timed out")


###########################################################
#### Запуск программы #####################################
###########################################################

# Эта часть программы запускается при старте,
# разбирает ввод пользователя, а затем запускает #
 желаемый режим программы.
if __name__ == "__main__":
    try:
        parser = argparse.ArgumentParser(description="Простой эхо-сервер и клиент␣
↪утилита")

        parser.add_argument(
            "-s",
            "--server",
            action="store_true",
            help="ждать входящие пакеты от клиентов"
        )

        parser.add_argument(
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            "-t",
            "--timeout",
            action="store",
            metavar="s",
            default=None,
            help="установить таймаут ответа в секундах",
            type=float
        )

        parser.add_argument("--config",
            action="store",
            default=None,
            help="путь к альтернативному каталогу конфигурации Reticulum",
            type=str
        )

        parser.add_argument(
            "destination",
            nargs="?",
            default=None,
            help="шестнадцатеричный хеш назначения сервера",
            type=str
        )

        args = parser.parse_args()

        if args.server:
            configarg=None
            if args.config:
                configarg = args.config
            server(configarg)
        else:
            if args.config:
                configarg = args.config
            else:
                configarg = None

            if args.timeout:
                timeoutarg = float(args.timeout)
            else:
                timeoutarg = None

            if (args.destination == None):
                print("")
                parser.print_help()
                print("")
            else:
                client(args.destination, configarg, timeout=timeoutarg)
    except KeyboardInterrupt:
        print("")
        sys.exit(0)
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/Echo.py.

9.5 Соединение

Пример *Link* демонстрирует установление зашифрованного соединения с удаленным пунктом назначения и передачу трафика в обоих направлениях по этому соединению.

```
############################################################
 # Этот пример RNS демонстрирует, как настроить соединение с #
# пунктом назначения и передавать данные туда и обратно по нему.  #
############################################################

import os
import sys
import time
import argparse
import RNS

# Давайте определим имя приложения. Мы будем использовать
это для всех # назначений, которые мы создаем. Поскольку
этот пример эха # является частью набора примеров утилит,
мы 'll поместим # их все в пространство имен приложения "
example_utilities" APP_NAME = "example_utilities"

############################################################
#### Серверная часть ######################################
############################################################

# Ссылка на последнее подключенное клиентское соединение
latest_client_link = None

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве сервера
def server(configpath):
    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Случайным образом создаем новую идентификацию для нашего примера соединения
    server_identity = RNS.Identity()

    # Мы создаем пункт назначения, к которому могут подключаться клиенты. Мы #
    хотим, чтобы клиенты создавали соединения с этим пунктом назначения, поэто-
    му нам # необходимо создать тип пункта назначения «единичный».
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.IN,
        RNS.Destination.SINGLE,
        APP_NAME,
        "linkexample"
    )

    # Мы настраиваем функцию, которая будет вызываться каждый раз,
    # когда новый клиент создает ссылку на это назначение.
    server_destination.set_link_established_callback(client_connected)
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    # Всё готово!
    # Пусть's Ждать запросов клиента или ввода пользователя.
    server_loop(server_destination)

def server_loop(destination):
    # Дайте пользователю знать, что всё готово
    RNS.log(
        "Пример ссылки "+
        RNS.prettyhexrep(destination.hash)+ " за-
        пущен, ожидается подключение."
    )

    RNS.log("Нажмите Enter для ручной отправки объявления (Ctrl-C для выхода)")

    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажимает Enter, мы объявим наше сер-
    верное # назначение в сети, что позволит клиентам # узнать,
     как создавать сообщения, направленные на него.
    while True:
        entered = input()
        destination.announce()
        RNS.log("Отправлено объявление от "+RNS.prettyhexrep(destination.hash))

# Когда клиент устанавливает связь с нашим серве-
ром # назначения, эта функция будет вызвана со #
ссылкой на связь.
def client_connected(link):
    global latest_client_link

    RNS.log("Клиент подключен")
    link.set_link_closed_callback(client_disconnected)
    link.set_packet_callback(server_packet_received)
    latest_client_link = link

def client_disconnected(link):
    RNS.log("Клиент отключен")

def server_packet_received(message, packet):
    global latest_client_link

    # Когда данные получены по любому активному каналу,
    # все они будут направлены последнему подключившемуся клиенту.
    # который подключен.
    text = message.decode("utf-8")
    RNS.log("Received data on the link: "+text)

    reply_text = "I received  \""+text+"\" over the link"
    reply_data = reply_text.encode("utf-8")
    RNS.Packet(latest_client_link, reply_data).send()


##########################################################
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
#### Часть клиента #########################################
############################################################

# Ссылка на соединение с сервером
server_link = None

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве клиента
def client(destination_hexhash, configpath):
    # Нам необходимо бинарное представление хеша назна-
    чения, # который был введен в командной строке
    try:
        dest_len = (RNS.Reticulum.TRUNCATED_HASHLENGTH//8)*2
        if len(destination_hexhash) != dest_len:
            raise ValueError(
                "Длина назначения недействительна, должно быть {hex} шестнадцатеричных символов (
↪{byte} байта).".format(hex=dest_len, byte=dest_len//2)
            )

        destination_hash = bytes.fromhex(destination_hexhash)
    except:
        RNS.log("Invalid destination entered. Check your input!\n")
        sys.exit(0)

    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Проверить, известен ли путь к назначению
    if not RNS.Transport.has_path(destination_hash):
        RNS.log("Destination is not yet known. Requesting path and waiting for announce␣
↪to arrive...")
        RNS.Transport.request_path(destination_hash)
        while not RNS.Transport.has_path(destination_hash):
            time.sleep(0.1)

    # Запоминаем идентификатор сервера
    server_identity = RNS.Identity.recall(destination_hash)

    # Информируем пользователя о начале подключения
    RNS.log("Установление связи с сервером...")

    # Когда идентификатор сервера известен, мы устанавливаем
    # назначение
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.OUT,
        RNS.Destination.SINGLE,
        APP_NAME,
        "linkexample"
    )

    # И создаем ссылку
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    link = RNS.Link(server_destination)

    # Мы устанавливаем колбэк, который будет выполнен
    # каждый раз, когда пакет будет получен через
    # ссылку
    link.set_packet_callback(client_packet_received)

    # Мы также настроим функции, чтобы информировать
    # пользователя, когда ссылка установлена или закрыта
    link.set_link_established_callback(link_established)
    link.set_link_closed_callback(link_closed)

    # Все настроено, так что ' давайте войдем в цикл #
    для взаимодействия пользователя с примером
    client_loop()

def client_loop():
    global server_link

    # Дождитесь, пока ссылка станет активной
    while not server_link:
        time.sleep(0.1)

    should_quit = False
    while not should_quit:
        try:
            print("> ", end=" ")
            text = input()

            # Проверьте, следует ли нам выйти из примера
            if text == "quit" or text == "q" or text == "exit":
                should_quit = True
                server_link.teardown()

            # Если нет, отправьте введенный текст по ссылке
            if text != "":
                data = text.encode("utf-8")
                if len(data) <= RNS.Link.MDU:
                    RNS.Packet(server_link, data).send()
                else:
                    RNS.log(
                        "Не удается отправить этот пакет, размер дан-
                        ных "+ str(len(data))+" байт превышает MDU пакета ссылки
                        "+ str(RNS.Link.MDU)+" байт", RNS.LOG_
                        ERROR
                    )

        except Exception as e:
            RNS.log("Error while sending data over the link: "+str(e))
            should_quit = True
            server_link.teardown()
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# Эта функция вызывается, когда связь #
 была установлена с сервером
def link_established(link):
    # Мы сохраняем ссылку на экземпляр # связи
    для дальнейшего использования
    global server_link
    server_link = link

    # Информируем пользователя о том,
    что сервер # подключен
    RNS.log("Link established with server, enter some text to send, or     \"quit\" to quit")

# Когда связь закрыта , мы'll информируем
# пользователя и выходим из программы
def link_closed(link):
    if link.teardown_reason == RNS.Link.TIMEOUT:
        RNS.log("The link timed out, exiting now")
    elif link.teardown_reason == RNS.Link.DESTINATION_CLOSED:
        RNS.log("The link was closed by the server, exiting now")
    else:
        RNS.log("Link closed, exiting now")

    time.sleep(1.5)
    sys.exit(0)

# Когда пакет получен по каналу, мы # просто
выводим данные.
def client_packet_received(message, packet):
    text = message.decode("utf-8") RNS.
    log("Полученные данные по каналу: "+text)
    print("> ", end=" ") sys.stdout.flush
    ()


############################################################
#### Запуск программы ######################################
############################################################

# Эта часть программы запускается при старте,
# и анализирует ввод пользователя, а затем
# запускает нужный режим программы.
if __name__ == "__main__":
    try:
        parser = argparse.ArgumentParser(description="Пример простой ссылки")

        parser.add_argument(
            "-s",
            "--server",
            action="store_true",
            help="ждать входящих запросов на соединение от клиентов"
        )
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        parser.add_argument(
            "--config",
            action="store",
            default=None,
            help="путь к альтернативному каталогу конфигурации Reticulum",
            type=str
        )

        parser.add_argument(
            "destination",
            nargs="?",
            default=None,
            help="шестнадцатеричный хеш назначения сервера",
            type=str
        )

        args = parser.parse_args()

        if args.config:
            configarg = args.config
        else:
            configarg = None

        if args.server:
            server(configarg)
        else:
            if (args.destination == None):
                print("")
                parser.print_help()
                print("")
            else:
                client(args.destination, configarg)

    except KeyboardInterrupt:
        print("")
        sys.exit(0)
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/Link.py.

9.6 Идентификация

Пример *Identify* исследует идентификацию инициатора соединения после его установления.

```
##########################################################
 # Этот пример RNS демонстрирует, как настроить соединение с #
# назначение и идентификация инициатора для его 's peer #
##########################################################

import os
import sys
import time
import argparse
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
import RNS

# Давайте определим имя приложения. Мы будем использовать
это для всех # назначений, которые мы создаем. Поскольку
этот пример эха # является частью набора примеров утилит,
мы'll поместим # их все в пространство имен приложения "
example_utilities" APP_NAME = "example_utilities"

############################################################
#### Серверная часть ######################################
############################################################

# Ссылка на последнее подключенное клиентское соединение
latest_client_link = None

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве сервера
def server(configpath):
    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Случайным образом создаем новую идентификацию для нашего примера соединения
    server_identity = RNS.Identity()

    # Мы создаем пункт назначения, к которому могут подключаться клиенты. Мы #
    хотим, чтобы клиенты создавали соединения с этим пунктом назначения, поэто-
    му нам # необходимо создать тип пункта назначения «единичный».
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.IN,
        RNS.Destination.SINGLE,
        APP_NAME,
        "identifyexample"
    )

    # Мы настраиваем функцию, которая будет вызываться каждый раз,
    # когда новый клиент создает ссылку на это назначение.
    server_destination.set_link_established_callback(client_connected)

    # Всё готово!
    # Пусть's Ждать запросов клиента или ввода пользователя.
    server_loop(server_destination)

def server_loop(destination):
    # Сообщаем пользователю, что все готово
    RNS.log(
        Пример идентификации канала "+ RNS.
        prettyhexrep(destination.hash)+ " запущен
        , ожидается подключение.
    )

    RNS.log("Нажмите Enter для ручной отправки объявления (Ctrl-C для выхода)")
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажимает Enter, мы объявим наше сер-
    верное # назначение в сети, что позволит клиентам # узнать,
     как создавать сообщения, направленные на него.
    while True:
        entered = input()
        destination.announce()
        RNS.log("Отправлено объявление от "+RNS.prettyhexrep(destination.hash))

# Когда клиент устанавливает связь с нашим серве-
ром # назначения, эта функция будет вызвана со #
ссылкой на связь.
def client_connected(link):
    global latest_client_link

    RNS.log("Клиент подключен")
    link.set_link_closed_callback(client_disconnected)
    link.set_packet_callback(server_packet_received)
    link.set_remote_identified_callback(remote_identified)
    latest_client_link = link

def client_disconnected(link):
    RNS.log("Клиент отключен")

def remote_identified(link, identity):
    RNS.log("Удаленный объект идентифицирован как: "+str(identity))

def server_packet_received(message, packet):
    global latest_client_link

    # Получаем исходный идентификатор для отображения
    remote_peer =  "unidentified peer"
    if packet.link.get_remote_identity() != None:
        remote_peer = str(packet.link.get_remote_identity())

    # Когда данные получены по любому активному каналу,
    # все они будут направлены последнему подключившемуся клиенту.
    # который подключен.
    text = message.decode("utf-8")

    RNS.log("Получены данные от "+remote_peer+": "+text)

    reply_text = "Я получил \" "+text+" \" по каналу от "+remote_peer reply_
    data = reply_text.encode("utf-8") RNS.
    Packet(latest_client_link, reply_data).send()


############################################################
#### Часть клиента #########################################
############################################################
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# Ссылка на соединение с сервером
server_link = None

# Ссылка на идентификатор клиента
client_identity = None

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве клиента
def client(destination_hexhash, configpath):
    global client_identity
    # Нам необходимо бинарное представление хеша назна-
    чения, # который был введен в командной строке
    try:
        dest_len = (RNS.Reticulum.TRUNCATED_HASHLENGTH//8)*2
        if len(destination_hexhash) != dest_len:
            raise ValueError(
                "Длина назначения недействительна, должно быть {hex} шестнадцатеричных символов (
↪{byte} байта)."".format(hex=dest_len, byte=dest_len//2)
            )

        destination_hash = bytes.fromhex(destination_hexhash)
    except:
        RNS.log("Invalid destination entered. Check your input!  \n")
        sys.exit(0)

    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Создание нового идентификатора клиента
    client_identity = RNS.Identity()
    RNS.log(
        "Клиент создал новый идентификатор "+
        str(client_identity)
    )

    # Проверить, известен ли путь к назначению
    if not RNS.Transport.has_path(destination_hash):
        RNS.log("Destination is not yet known. Requesting path and waiting for announce␣
↪to arrive...")
        RNS.Transport.request_path(destination_hash)
        while not RNS.Transport.has_path(destination_hash):
            time.sleep(0.1)

    # Запоминаем идентификатор сервера
    server_identity = RNS.Identity.recall(destination_hash)

    # Информируем пользователя о начале подключения
    RNS.log("Установление связи с сервером...")

    # Когда идентификатор сервера известен, мы устанавливаем
    # назначение
    server_destination = RNS.Destination(
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        server_identity,
        RNS.Destination.OUT,
        RNS.Destination.SINGLE,
        APP_NAME,
        "identifyexample"
    )

    # И создаем ссылку
    link = RNS.Link(server_destination)

    # Мы устанавливаем колбэк, который будет выполнен
    # каждый раз, когда пакет будет получен через
    # ссылку
    link.set_packet_callback(client_packet_received)

    # Мы также настроим функции, чтобы информировать
    # пользователя, когда ссылка установлена или закрыта
    link.set_link_established_callback(link_established)
    link.set_link_closed_callback(link_closed)

    # Все настроено, так что ' давайте войдем в цикл #
    для взаимодействия пользователя с примером
    client_loop()

def client_loop():
    global server_link

    # Дождитесь, пока ссылка станет активной
    while not server_link:
        time.sleep(0.1)

    should_quit = False
    while not should_quit:
        try:
            print("> ", end=" ")
            text = input()

            # Проверьте, следует ли нам выйти из примера
            if text == "quit" or text == "q" or text == "exit":
                should_quit = True
                server_link.teardown()

            # Если нет, отправьте введенный текст по ссылке
            if text != "":
                data = text.encode("utf-8")
                if len(data) <= RNS.Link.MDU:
                    RNS.Packet(server_link, data).send()
                else:
                    RNS.log(
                        "Невозможно отправить этот пакет, размер данных "+ str(len(data
                        ))+" байтов превышает MDU пакета канала "+ str(RNS.Link.
                        MDU)+" байтов",
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
                        RNS.LOG_ERROR
                    )

        except Exception as e:
            RNS.log("Error while sending data over the link: "+str(e))
            should_quit = True
            server_link.teardown()

# Эта функция вызывается, когда связь #
 была установлена с сервером
def link_established(link):
    # Мы сохраняем ссылку на экземпляр # связи
    для дальнейшего использования
    global server_link, client_identity
    server_link = link

    # Информируем пользователя о том,
    что сервер # подключен
    RNS.log("Установлено соединение с сервером, идентификация удаленного узла...")

    link.identify(client_identity)

# Когда связь закрыта , мы'll информируем
# пользователя и выходим из программы
def link_closed(link):
    if link.teardown_reason == RNS.Link.TIMEOUT:
        RNS.log("The link timed out, exiting now")
    elif link.teardown_reason == RNS.Link.DESTINATION_CLOSED:
        RNS.log("The link was closed by the server, exiting now")
    else:
        RNS.log("Link closed, exiting now")

    time.sleep(1.5)
    sys.exit(0)

# Когда пакет получен по каналу, мы # просто
выводим данные.
def client_packet_received(message, packet):
    text = message.decode("utf-8") RNS.
    log("Полученные данные по каналу: "+text)
    print("> ", end=" ") sys.stdout.flush
    ()


############################################################
#### Запуск программы ######################################
############################################################

# Эта часть программы запускается при старте,
# и анализирует ввод пользователя, а затем
# запускает нужный режим программы.
if __name__ == "__main__":
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    try:
        parser = argparse.ArgumentParser(description="Пример простой ссылки")

        parser.add_argument(
            "-s",
            "--server",
            action="store_true",
            help="ждать входящих запросов на соединение от клиентов"
        )

        parser.add_argument(
            "--config",
            action="store",
            default=None,
            help="путь к альтернативному каталогу конфигурации Reticulum",
            type=str
        )

        parser.add_argument(
            "destination",
            nargs="?",
            default=None,
            help="шестнадцатеричный хеш назначения сервера",
            type=str
        )

        args = parser.parse_args()

        if args.config:
            configarg = args.config
        else:
            configarg = None

        if args.server:
            server(configarg)
        else:
            if (args.destination == None):
                print("")
                parser.print_help()
                print("")
            else:
                client(args.destination, configarg)

    except KeyboardInterrupt:
        print("")
        sys.exit(0)
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/Identify.py.

9.7 Запросы и ответы

Пример *Request*исследует отправку запросов и получение ответов.

```
##########################################################
 # Этот пример RNS демонстрирует, как выполнять запросы #
# и получать ответы по ссылке.                           #
##########################################################

import os
import sys
import time
import random
import argparse
import RNS

# Давайте определим имя приложения. Мы будем использовать
это для всех # назначений, которые мы создаем. Поскольку
этот пример эха # является частью набора примеров утилит,
мы 'll поместим # их все в пространство имен приложения "
example_utilities" APP_NAME = "example_utilities"

##########################################################
#### Серверная часть #####################################
##########################################################

# Ссылка на последнее подключенное клиентское соединение
latest_client_link = None

def random_text_generator(path, data, request_id, link_id, remote_identity, requested_
↪at):
    RNS.log("Генерация ответа на запрос "+RNS.prettyhexrep(request_id)+" по ссылке
↪"+RNS.prettyhexrep(link_id))
    texts = ["They looked up", "On each full moon", "Becky was upset", "I'll stay away␣
↪from it", "The pet shop stocks everything"]
    return texts[random.randint(0, len(texts)-1)]

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве сервера
def server(configpath):
    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Случайным образом создаем новую идентификацию для нашего примера соединения
    server_identity = RNS.Identity()

    # Мы создаем пункт назначения, к которому могут подключаться клиенты. Мы #
    хотим, чтобы клиенты создавали соединения с этим пунктом назначения, поэто-
    му нам # необходимо создать тип пункта назначения «единичный».
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.IN,
        RNS.Destination.SINGLE,
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        APP_NAME,
        "requestexample"
    )

    # Мы настраиваем функцию, которая будет вызываться каждый раз,
    # когда новый клиент создает ссылку на это назначение.
    server_destination.set_link_established_callback(client_connected)

    # Мы регистрируем обработчик запросов для обработки вхо-
    дящих # запросов по любым установленным ссылкам.
    server_destination.register_request_handler(
        "/random/text",
        response_generator = random_text_generator,
        allow = RNS.Destination.ALLOW_ALL
    )

    # Всё готово!
    # Пусть's Ждать запросов клиента или ввода пользователя.
    server_loop(server_destination)

def server_loop(destination):
    # Дайте пользователю знать, что всё готово
    RNS.log(
        "Пример запроса "+
        RNS.prettyhexrep(destination.hash)+ " за-
        пущен, ожидается подключение."
    )

    RNS.log("Нажмите Enter для ручной отправки объявления (Ctrl-C для выхода)")

    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажимает Enter, мы объявим наше сер-
    верное # назначение в сети, что позволит клиентам # узнать,
     как создавать сообщения, направленные на него.
    while True:
        entered = input()
        destination.announce()
        RNS.log("Отправлено объявление от "+RNS.prettyhexrep(destination.hash))

# Когда клиент устанавливает связь с нашим серве-
ром # назначения, эта функция будет вызвана со #
ссылкой на связь.
def client_connected(link):
    global latest_client_link

    RNS.log("Клиент подключен")
    link.set_link_closed_callback(client_disconnected)
    latest_client_link = link

def client_disconnected(link):
    RNS.log("Клиент отключен")
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
############################################################
#### Часть клиента #########################################
############################################################

# Ссылка на соединение с сервером
server_link = None

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве клиента
def client(destination_hexhash, configpath):
    # Нам необходимо бинарное представление хеша назна-
    чения, # который был введен в командной строке
    try:
        dest_len = (RNS.Reticulum.TRUNCATED_HASHLENGTH//8)*2
        if len(destination_hexhash) != dest_len:
            raise ValueError(
                "Длина назначения недействительна, должно быть {hex} шестнадцатеричных символов (
↪{byte} байта).".format(hex=dest_len, byte=dest_len//2)
            )

        destination_hash = bytes.fromhex(destination_hexhash)
    except:
        RNS.log("Invalid destination entered. Check your input!\n")
        sys.exit(0)

    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Проверить, известен ли путь к назначению
    if not RNS.Transport.has_path(destination_hash):
        RNS.log("Destination is not yet known. Requesting path and waiting for announce␣
↪to arrive...")
        RNS.Transport.request_path(destination_hash)
        while not RNS.Transport.has_path(destination_hash):
            time.sleep(0.1)

    # Запоминаем идентификатор сервера
    server_identity = RNS.Identity.recall(destination_hash)

    # Информируем пользователя о начале подключения
    RNS.log("Установление связи с сервером...")

    # Когда идентификатор сервера известен, мы устанавливаем
    # назначение
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.OUT,
        RNS.Destination.SINGLE,
        APP_NAME,
        "requestexample"
    )
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    # И создаем ссылку
    link = RNS.Link(server_destination)

    # Мы настроим функции для информирования # пользова-
    теля, когда соединение установлено или закрыто
    link.set_link_established_callback(link_established)
    link.set_link_closed_callback(link_closed)

    # Все настроено, так что ' давайте войдем в цикл #
    для взаимодействия пользователя с примером
    client_loop()

def client_loop():
    global server_link

    # Дождитесь, пока ссылка станет активной
    while not server_link:
        time.sleep(0.1)

    should_quit = False
    while not should_quit:
        try:
            print("> ", end=" ")
            text = input()

            # Проверьте, следует ли нам выйти из примера
            if text == "quit" or text == "q" or text == "exit":
                should_quit = True
                server_link.teardown()

            else:
                server_link.request(
                    "/random/text",
                    data = None,
                    response_callback = got_response,
                    failed_callback = request_failed
                )


        except Exception as e:
            RNS.log("Ошибка при отправке запроса по ссылке: "+str(e)) should_quit =
             True server_link.
            teardown()

def got_response(request_receipt):
    request_id = request_receipt.request_id
    response = request_receipt.response

    RNS.log("Получен ответ на запрос "+RNS.prettyhexrep(request_id)+": "+str(response))

def request_received(request_receipt):
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    RNS.log("Запрос "+RNS.prettyhexrep(request_receipt.request_id)+" был получен␣
↪удаленным пиром.")

def request_failed(request_receipt):
    RNS.log("Запрос "+RNS.prettyhexrep(request_receipt.request_id)+" не выполнен.")


# Эта функция вызывается, когда связь #
 была установлена с сервером
def link_established(link):
    # Мы сохраняем ссылку на экземпляр # связи
    для дальнейшего использования
    global server_link
    server_link = link

    # Информируем пользователя о том,
    что сервер # подключен
    RNS.log("Link established with server, hit enter to perform a request, or type in      \
↪"quit\" чтобы выйти")

# Когда связь закрыта , мы'll информируем
# пользователя и выходим из программы
def link_closed(link):
    if link.teardown_reason == RNS.Link.TIMEOUT:
        RNS.log("The link timed out, exiting now")
    elif link.teardown_reason == RNS.Link.DESTINATION_CLOSED:
        RNS.log("The link was closed by the server, exiting now")
    else:
        RNS.log("Link closed, exiting now")

    time.sleep(1.5)
    sys.exit(0)


############################################################
#### Запуск программы ######################################
############################################################

# Эта часть программы запускается при старте,
# и анализирует ввод пользователя, а затем
# запускает нужный режим программы.
if __name__ == "__main__":
    try:
        parser = argparse.ArgumentParser(description="Простой пример запрос/ответ")

        parser.add_argument(
            "-s",
            "--server",
            action="store_true",
            help="ждать входящих запросов от клиентов"
        )
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        parser.add_argument(
            "--config",
            action="store",
            default=None,
            help="путь к альтернативному каталогу конфигурации Reticulum",
            type=str
        )

        parser.add_argument(
            "destination",
            nargs="?",
            default=None,
            help="шестнадцатеричный хеш назначения сервера",
            type=str
        )

        args = parser.parse_args()

        if args.config:
            configarg = args.config
        else:
            configarg = None

        if args.server:
            server(configarg)
        else:
            if (args.destination == None):
                print("")
                parser.print_help()
                print("")
            else:
                client(args.destination, configarg)

    except KeyboardInterrupt:
        print("")
        sys.exit(0)
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/Request.py.

9.8 Канал

Пример *Channel*исследует использование Channelдля отправки структурированных данных между узлами Link.

```
##########################################################
 # Этот пример RNS демонстрирует, как настроить соединение с #
# место назначения и передавать по нему структурированные сообщения  #
# используя канал.                                       #
##########################################################

import os
import sys
import time
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
import argparse
from datetime import  datetime

import RNS
from RNS.vendor import  umsgpack

# Давайте определим имя приложения. Мы будем использовать
это для всех # назначений, которые мы создаем. Поскольку
этот пример эха # является частью набора примеров утилит,
мы 'll поместим # их все в пространство имен приложения "
example_utilities" APP_NAME = "example_utilities"

############################################################
#### Общие объекты #########################################
############################################################

# Данные канала должны быть структурированы в подклассе
# MessageBase. Это гарантирует, что канал сможет # сериа-
лизовать и десериализовать объект и мультиплексировать
его # с другими объектами. Оба конца канала должны иметь #
 одинаковые определения объектов, чтобы иметь возмож-
ность обмениваться данными по # каналу.
#
# Примечание: Объекты, которые мы хотим использовать в ка-
нале, должны быть зарегистрированы в канале, и каждое соеди-
нение имеет # свой собственный экземпляр канала. См. функции
client_connected # и link_established в этом примере, чтобы
увидеть, # как регистрируются типы сообщений.

# Давайте создадим простой класс сообщений под названием StringMessage
# который будет передавать строку с отметкой времени.

class StringMessage(RNS.MessageBase):
    # Переменной класса MSGTYPE необходимо присвоить # 2-
    байтовое целочисленное значение. Этот идентификатор
    позволяет # каналу найти конструктор вашего сообщения
    , когда # сообщение поступает по каналу.
    #
    # MSGTYPE должен быть уникальным для всех типов сооб-
    щений, которые мы # регистрируем в канале. MSGTYPEs >
    = 0xf000 # зарезервированы для системы.
    MSGTYPE = 0x0101

    # Конструктор нашего объекта должен быть вызываемым
    без # аргументов. Мы можем иметь параметры, но они
    должны # иметь присвоение по умолчанию.
    #
    # Это необходимо, чтобы канал мог создать пустую
    # версию нашего сообщения, в которую входящее
    # сообщение может быть распаковано.
    def __init__(self, data=None):
        self.data = data
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        self.timestamp = datetime.now()

    # Наконец, наше сообщение должно реализовывать функции #, которые
    канал может вызывать для упаковки и распаковки нашего сообщения # в/
    из полезной нагрузки необработанного пакета. Мы ' будем использовать
    пакет # umsgpack, поставляемый с RNS. Мы также могли бы использо-
    вать # пакет struct, поставляемый с Python, если бы хотели # больше
    контроля над структурой упакованных байтов.
    #
    # Также обратите внимание, что упакованные объекты со-
    общений должны полностью # помещаться в один пакет.
    Количество байтов #, доступных для полезных нагрузок со-
    общений, можно запросить у # канала с помощью свойства
    Channel.MDU. MDU # канала немного меньше MDU # ссылки из-за
    кодирования заголовка сообщения.

    # Функция pack кодирует содержимое сообщения в # поток
    байтов.
    def pack(self) -> bytes:
        return umsgpack.packb((self.data, self.timestamp))

    # И функция unpack декодирует поток байтов в
    # содержимое сообщения.
    def unpack(self, raw):
        self.data, self.timestamp = umsgpack.unpackb(raw)


############################################################
#### Серверная часть ######################################
############################################################

# Ссылка на последнее подключенное клиентское соединение
latest_client_link = None

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве сервера
def server(configpath):
    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Случайным образом создаем новую идентификацию для нашего примера соединения
    server_identity = RNS.Identity()

    # Мы создаем пункт назначения, к которому могут подключаться клиенты. Мы #
    хотим, чтобы клиенты создавали соединения с этим пунктом назначения, поэто-
    му нам # необходимо создать тип пункта назначения «единичный».
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.IN,
        RNS.Destination.SINGLE,
        APP_NAME,
        "channelexample"
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    )

    # Мы настраиваем функцию, которая будет вызываться каждый раз,
    # когда новый клиент создает ссылку на это назначение.
    server_destination.set_link_established_callback(client_connected)

    # Всё готово!
    # Пусть's Ждать запросов клиента или ввода пользователя.
    server_loop(server_destination)

def server_loop(destination):
    # Дайте пользователю знать, что всё готово
    RNS.log(
        "Пример канала "+
        RNS.prettyhexrep(destination.hash)+ " за-
        пущен, ожидается подключение."
    )

    RNS.log("Нажмите Enter для ручной отправки объявления (Ctrl-C для выхода)")

    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажимает Enter, мы объявим наше сер-
    верное # назначение в сети, что позволит клиентам # узнать,
     как создавать сообщения, направленные на него.
    while True:
        entered = input()
        destination.announce()
        RNS.log("Отправлено объявление от "+RNS.prettyhexrep(destination.hash))

# Когда клиент устанавливает связь с нашим серве-
ром # назначения, эта функция будет вызвана со #
ссылкой на связь.
def client_connected(link):
    global latest_client_link
    latest_client_link = link

    RNS.log("Клиент подключен")
    link.set_link_closed_callback(client_disconnected)

    # Зарегистрировать типы сообщений и добавить обработчик к каналу
    channel = link.get_channel()
    channel.register_message_type(StringMessage)
    channel.add_message_handler(server_message_received)

def client_disconnected(link):
    RNS.log("Клиент отключен")

def server_message_received(message):
    """
    Обработчик сообщений
    @param message: Экземпляр подкласса MessageBase
    @return: True, если сообщение было обработано
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    """
    global latest_client_link
    # Когда сообщение получено по любой активной ссылке,
    # все ответы будут направлены последнему клиенту
    # который подключен.

    # В обработчике сообщений любое десериализуемое сообщение,
    # которое поступает по каналу 's, будет передано
    # всем обработчикам сообщений, если предыдущий обработчик не укажет, что он
    # обработал сообщение.
    #
    #
    if isinstance(message, StringMessage):
        RNS.log("Получены данные по каналу: " + message.data + " (сообщение создано в " +␣
↪str(message.timestamp) + ")")

        reply_message = StringMessage("Я получил \" "+message.data+" \" по каналу")
        latest_client_link.get_channel().send(reply_message)

        # Входящие сообщения отправляются каждому обра-
        ботчику # сообщений, добавленному в канал, в том
        порядке, в котором они # были добавлены.
        # Если какой-либо обработчик сообщений возвращает
        True, сообщение # считается обработанным, и все по-
        следующие # обработчики пропускаются.
        return True


############################################################
#### Часть клиента #########################################
############################################################

# Ссылка на соединение с сервером
server_link = None

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве клиента
def client(destination_hexhash, configpath):
    # Нам необходимо бинарное представление хеша назна-
    чения, # который был введен в командной строке
    try:
        dest_len = (RNS.Reticulum.TRUNCATED_HASHLENGTH//8)*2
        if len(destination_hexhash) != dest_len:
            raise ValueError(
                "Длина назначения недействительна, должно быть {hex} шестнадцатеричных символов (
↪{byte} байта).".format(hex=dest_len, byte=dest_len//2)
            )

        destination_hash = bytes.fromhex(destination_hexhash)
    except:
        RNS.log("Invalid destination entered. Check your input! \n")
        sys.exit(0)
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Проверить, известен ли путь к назначению
    if not RNS.Transport.has_path(destination_hash):
        RNS.log("Destination is not yet known. Requesting path and waiting for announce␣
↪to arrive...")
        RNS.Transport.request_path(destination_hash)
        while not RNS.Transport.has_path(destination_hash):
            time.sleep(0.1)

    # Запоминаем идентификатор сервера
    server_identity = RNS.Identity.recall(destination_hash)

    # Информируем пользователя о начале подключения
    RNS.log("Установление связи с сервером...")

    # Когда идентификатор сервера известен, мы устанавливаем
    # назначение
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.OUT,
        RNS.Destination.SINGLE,
        APP_NAME,
        "channelexample"
    )

    # И создаем ссылку
    link = RNS.Link(server_destination)

    # Мы также настроим функции, чтобы информировать
    # пользователя, когда ссылка установлена или закрыта
    link.set_link_established_callback(link_established)
    link.set_link_closed_callback(link_closed)

    # Все настроено, так что ' давайте войдем в цикл #
    для взаимодействия пользователя с примером
    client_loop()

def client_loop():
    global server_link

    # Дождитесь, пока ссылка станет активной
    while not server_link:
        time.sleep(0.1)

    should_quit = False
    while not should_quit:
        try:
            print("> ", end=" ")
            text = input()
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            # Проверьте, следует ли нам выйти из примера
            if text == "quit" or text == "q" or text == "exit":
                should_quit = True
                server_link.teardown()

            # Если нет, отправьте введенный текст по ссылке
            if text != "":
                message = StringMessage(text)
                packed_size = len(message.pack())
                channel = server_link.get_channel()
                if channel.is_ready_to_send():
                    if packed_size <= channel.mdu:
                        channel.send(message)
                    else:
                        RNS.log(
                            "Невозможно отправить этот пакет, размер данных "+ str(
                            packed_size)+" байт превышает MDU пакета канала "+ str(
                            channel.MDU)+" байт", RNS.
                            LOG_ERROR
                        )
                else:
                    RNS.log("Канал не готов к отправке, пожалуйста, дождитесь заверше-
                            ния ожидающих сообщений.", RNS.LOG_ERROR)

        except Exception as e:
            RNS.log("Error while sending data over the link: "+str(e))
            should_quit = True
            server_link.teardown()

# Эта функция вызывается, когда связь #
 была установлена с сервером
def link_established(link):
    # Мы сохраняем ссылку на экземпляр # связи
    для дальнейшего использования
    global server_link
    server_link = link

    # Зарегистрировать сообщения и добавить обработчик в канал
    channel = link.get_channel()
    channel.register_message_type(StringMessage)
    channel.add_message_handler(client_message_received)

    # Информируем пользователя о том,
    что сервер # подключен
    RNS.log("Link established with server, enter some text to send, or     \"quit\" to quit")

# Когда связь закрыта , мы'll информируем
# пользователя и выходим из программы
def link_closed(link):
    if link.teardown_reason == RNS.Link.TIMEOUT:
        RNS.log("Время ожидания канала истекло, выход")
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    elif link.teardown_reason == RNS.Link.DESTINATION_CLOSED:
        RNS.log("The link was closed by the server, exiting now")
    else:
        RNS.log("Link closed, exiting now")

    time.sleep(1.5)
    sys.exit(0)

# Когда пакет получен по каналу, мы
# просто выводим данные.
def client_message_received(message):
    if isinstance(message, StringMessage):
        RNS.log("Получены данные по каналу: " + message.data + " (сообщение создано в " +␣
↪str(message.timestamp) + ")")
        print("> ", end=" ")
        sys.stdout.flush()


############################################################
#### Запуск программы ######################################
############################################################

# Эта часть программы запускается при старте,
# и анализирует ввод пользователя, а затем
# запускает нужный режим программы.
if __name__ == "__main__":
    try:
        parser = argparse.ArgumentParser(description="Simple channel example")

        parser.add_argument(
            "-s",
            "--server",
            action="store_true",
            help="ждать входящих запросов на соединение от клиентов"
        )

        parser.add_argument(
            "--config",
            action="store",
            default=None,
            help="путь к альтернативному каталогу конфигурации Reticulum",
            type=str
        )

        parser.add_argument(
            "destination",
            nargs="?",
            default=None,
            help="шестнадцатеричный хеш назначения сервера",
            type=str
        )
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        args = parser.parse_args()

        if args.config:
            configarg = args.config
        else:
            configarg = None

        if args.server:
            server(configarg)
        else:
            if (args.destination == None):
                print("")
                parser.print_help()
                print("")
            else:
                client(args.destination, configarg)

    except KeyboardInterrupt:
        print("")
        sys.exit(0)
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/Channel.py.

9.9 Буфер

Пример *Buffer*исследует использование буферизованных читателей и писателей для отправки бинарных данных между одноранговыми узлами `Link`.

```
##########################################################
 # Этот пример RNS демонстрирует, как настроить соединение с #
# назначение, и передать бинарные данные через него с помощью  #
# буфер канала.                                            #
##########################################################
from __future__ import  annotations
import os
import sys
import time
import argparse
from datetime import  datetime

import RNS
from RNS.vendor import  umsgpack

# Давайте определим имя приложения. Мы будем использовать
это для всех # назначений, которые мы создаем. Поскольку
этот пример эха # является частью набора примеров утилит,
мы 'll поместим # их все в пространство имен приложения "
example_utilities" APP_NAME = "example_utilities"


##########################################################
#### Серверная часть ###################################
##########################################################
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# Ссылка на последнее подключенное клиентское соединение
latest_client_link = None

# Ссылка на последний объект буфера
latest_buffer = None

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве сервера
def server(configpath):
    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Случайно создадим новую личность для нашего примера
    server_identity = RNS.Identity()

    # Мы создаем пункт назначения, к которому могут подключаться клиенты. Мы #
    хотим, чтобы клиенты создавали соединения с этим пунктом назначения, поэто-
    му нам # необходимо создать тип пункта назначения «единичный».
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.IN,
        RNS.Destination.SINGLE,
        APP_NAME,
        "bufferexample"
    )

    # Мы настраиваем функцию, которая будет вызываться каждый раз,
    # когда новый клиент создает ссылку на это назначение.
    server_destination.set_link_established_callback(client_connected)

    # Всё готово!
    # Пусть's Ждать запросов клиента или ввода пользователя.
    server_loop(server_destination)

def server_loop(destination):
    # Сообщаем пользователю, что все готово
    RNS.log(
        "Пример буфера ссылки "+ RNS.
        prettyhexrep(destination.hash)+ " за-
        пущен, ожидается соединение."
    )

    RNS.log("Нажмите Enter для ручной отправки объявления (Ctrl-C для выхода)")

    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажимает Enter, мы объявим наше сер-
    верное # назначение в сети, что позволит клиентам # узнать,
     как создавать сообщения, направленные на него.
    while True:
        entered = input()
        destination.announce()
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        RNS.log("Отправлено объявление от "+RNS.prettyhexrep(destination.hash))

# Когда клиент устанавливает связь с нашим серве-
ром # назначения, эта функция будет вызвана со #
ссылкой на связь.
def client_connected(link):
    global latest_client_link, latest_buffer
    latest_client_link = link

    RNS.log("Клиент подключен")
    link.set_link_closed_callback(client_disconnected)

    # Если получено новое соединение, старый ридер
    # необходимо отключить.
    if latest_buffer:
        latest_buffer.close()


    # Создать буферные объекты.
    #   Параметр stream_id этих функций
    #   немного похож на файловый дескриптор, за исключением того, что он
    #   уникален для *получателя*.
    #
    #   В этом примере как читатель, так и писатель
    #   используют stream_id = 0, но на самом деле существуют два
    #   отдельных однонаправленных потока, текущих в
    #   противоположных направлениях.
    #
    channel = link.get_channel()
    latest_buffer = RNS.Buffer.create_bidirectional_buffer(0, 0, channel, server_buffer_
↪ready)

def client_disconnected(link):
    RNS.log("Клиент отключен")

def server_buffer_ready(ready_bytes: int):
    """
    Обратный вызов из буфера, когда в буфере доступны данные

    :param ready_bytes: Количество байтов, готовых к чтению
    """
    global latest_buffer

    data = latest_buffer.read(ready_bytes)
    data = data.decode("utf-8")

    RNS.log("Получены данные через буфер: " + data)

    reply_message = "Я получил  \" "+data+" \" через буфер" reply
    _message = reply_message.encode("utf-8") latest
    _buffer.write(reply_message) latest_buffer.
    flush()
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
##########################################################
#### Часть клиента #######################################
##########################################################

# Ссылка на соединение с сервером
server_link = None

# Ссылка на объект буфера, необходимая для совместного использования
# объекта из обратного вызова подключенного канала связи
# клиентскому циклу.
buffer = None

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве клиента
def client(destination_hexhash, configpath):
    # Нам необходимо бинарное представление хеша назна-
    чения, # который был введен в командной строке
    try:
        dest_len = (RNS.Reticulum.TRUNCATED_HASHLENGTH//8)*2
        if len(destination_hexhash) != dest_len:
            raise ValueError(
                "Длина назначения недействительна, должно быть {hex} шестнадцатеричных символов (
↪{byte} байта).".format(hex=dest_len, byte=dest_len//2)
            )

        destination_hash = bytes.fromhex(destination_hexhash)
    except:
        RNS.log("Invalid destination entered. Check your input!\n")
        sys.exit(0)

    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Проверить, известен ли путь к назначению
    if not RNS.Transport.has_path(destination_hash):
        RNS.log("Destination is not yet known. Requesting path and waiting for announce␣
↪to arrive...")
        RNS.Transport.request_path(destination_hash)
        while not RNS.Transport.has_path(destination_hash):
            time.sleep(0.1)

    # Запоминаем идентификатор сервера
    server_identity = RNS.Identity.recall(destination_hash)

    # Информируем пользователя о начале подключения
    RNS.log("Установление связи с сервером...")

    # Когда идентификатор сервера известен, мы устанавливаем
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    # назначение.
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.OUT,
        RNS.Destination.SINGLE,
        APP_NAME,
        "bufferexample"
    )

    # И создаем ссылку
    link = RNS.Link(server_destination)

    # Мы также настроим функции, чтобы информировать
    # пользователя, когда ссылка установлена или закрыта
    link.set_link_established_callback(link_established)
    link.set_link_closed_callback(link_closed)

    # Все настроено, так что ' давайте войдем в цикл #
    для взаимодействия пользователя с примером
    client_loop()

def client_loop():
    global server_link

    # Дождитесь, пока ссылка станет активной
    while not server_link:
        time.sleep(0.1)

    should_quit = False
    while not should_quit:
        try:
            print("> ", end=" ")
            text = input()

            # Проверьте, следует ли нам выйти из примера
            if text == "quit" or text == "q" or text == "exit":
                should_quit = True
                server_link.teardown()
            else:
                # В противном случае, закодируйте текст и запишите его в буфер.
                text = text.encode("utf-8")
                buffer.write(text)
                # Очистите буфер, чтобы принудительно отправить данные.
                buffer.flush()


        except Exception as e:
            RNS.log("Ошибка при отправке данных через буфер канала: "+str(e)) should_
            quit = True server_
            link.teardown()

# Эта функция вызывается, когда соединение
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# установлено с сервером
def link_established(link):
    # Мы сохраняем ссылку на экземпляр
    # соединения для дальнейшего использования
    global server_link, buffer
    server_link = link

    # Создайте буфер; подробнее о настройке буфера см.
    в server_client_connected().
    channel = link.get_channel()
    buffer = RNS.Buffer.create_bidirectional_buffer(0, 0, channel, client_buffer_ready)

    # Информируем пользователя о том,
    что сервер # подключен
    RNS.log("Link established with server, enter some text to send, or     \"quit\" to quit")

# Когда связь закрыта , мы'll информируем
# пользователя и выходим из программы
def link_closed(link):
    if link.teardown_reason == RNS.Link.TIMEOUT:
        RNS.log("The link timed out, exiting now")
    elif link.teardown_reason == RNS.Link.DESTINATION_CLOSED:
        RNS.log("The link was closed by the server, exiting now")
    else:
        RNS.log("Link closed, exiting now")

    time.sleep(1.5)
    sys.exit(0)

# Когда в буфере появляются новые данные , считайте их и запишите в терминал.
def client_buffer_ready(ready_bytes: int):
    global buffer
    data = buffer.read(ready_bytes)
    RNS.log("Received data over the link buffer: " + data.decode("utf-8"))
    print("> ", end=" ")
    sys.stdout.flush()


##########################################################
#### Запуск программы ####################################
##########################################################

# Эта часть программы запускается при старте ,
# и анализирует ввод пользователя , а затем
# запускает нужный режим программы.
if __name__ == "__main__":
    try:
        parser = argparse.ArgumentParser(description="Простой пример буфера")

        parser.add_argument(
            "-s",
            "--server",
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            action="store_true",
            help="ждать входящих запросов на соединение от клиентов"
        )

        parser.add_argument(
            "--config",
            action="store",
            default=None,
            help="путь к альтернативному каталогу конфигурации Reticulum",
            type=str
        )

        parser.add_argument(
            "destination",
            nargs="?",
            default=None,
            help="шестнадцатеричный хеш назначения сервера",
            type=str
        )

        args = parser.parse_args()

        if args.config:
            configarg = args.config
        else:
            configarg = None

        if args.server:
            server(configarg)
        else:
            if (args.destination == None):
                print("")
                parser.print_help()
                print("")
            else:
                client(args.destination, configarg)

    except KeyboardInterrupt:
        print("")
        sys.exit(0)
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/Buffer.py.

9.10 Передача файлов

Пример *Filetransfer* реализует базовую программу файлового сервера, которая позволяет клиентам подключаться и загружать файлы. Программа использует интерфейс Resource для эффективной передачи файлов любого размера через Reticulum *Link* .

```
##########################################################
Этот пример RNS демонстрирует простую программу файлового   # #
 сервера и клиента. Сервер будет обслуживать   # # каталог файлов,
 а клиенты могут выводить список и                          #
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# загружать файлы с сервера.                                    #
#                                                               # # Обратите вни-
мание, что использование ресурсов RNS для передачи больших файлов  # # не ре-
комендуется, поскольку сжатие,                                  #
# шифрование и последовательность хэш-карт могут занимать много времени #
 # на системах с медленными процессорами, что, вероятно, приведет #
# к истечению времени ожидания клиента до того, как отправитель ресурса  #
# сможет завершить подготовку ресурса.                          #
#                                                               #
# Если вам нужно передавать большие файлы, используйте вместо этого класс Bundle  #, который автоматически нарезает данные  #
# класса вместо этого, который будет автоматически нарезать данные #
# на фрагменты, подходящие для упаковки в качестве ресурса.     #
################################################################

import os
import sys
import time
import threading
import argparse
import RNS
import RNS.vendor.umsgpack as umsgpack

# Давайте определим имя приложения. Мы будем использовать
это для всех # назначений, которые мы создаем. Поскольку
этот пример эха # является частью набора примеров утилит,
мы 'll поместим # их все в пространство имен приложения "
example_utilities" APP_NAME = "example_utilities"

# Мы также определим тайм-аут по умолчанию, в секундах
APP_TIMEOUT = 45.0

################################################################
#### Серверная часть ###########################################
################################################################

serve_path = None

# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве сервера
def server(configpath, path):
    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)

    # Случайным образом создаем новую идентификацию для нашего файлового сервера
    server_identity = RNS.Identity()

    global serve_path
    serve_path = path

    # Мы создаем пункт назначения, к которому могут подключаться клиенты. Мы #
    хотим, чтобы клиенты создавали соединения с этим пунктом назначения, поэто-
    му нам # необходимо создать тип пункта назначения «единичный».
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.IN,
        RNS.Destination.SINGLE,
        APP_NAME,
        "filetransfer",
        "server"
    )

    # Мы настраиваем функцию, которая будет вызываться каждый раз,
    # когда новый клиент создает ссылку на это назначение.
    server_destination.set_link_established_callback(client_connected)

    # Всё готово!
    # Пусть's Ждать запросов клиента или ввода пользователя.
    announceLoop(server_destination)

def announceLoop(destination):
    # Дайте пользователю знать, что всё готово
    RNS.log("Файловый сервер "+RNS.prettyhexrep(destination.hash)+" запущен") RNS.log
    ("Нажмите Enter, чтобы вручную отправить объявление (Ctrl-C для выхода)")

    # Мы входим в цикл, который выполняется до тех пор, пока пользователь не выйдет.
    # Если пользователь нажимает Enter, мы объявим наше сер-
    верное # назначение в сети, что позволит клиентам # узнать,
     как создавать сообщения, направленные на него.
    while True:
        entered = input()
        destination.announce()
        RNS.log("Отправлено объявление от "+RNS.prettyhexrep(destination.hash))

# Вот удобная функция для вывода списка всех файлов #
в нашем обслуживаемом каталоге
def list_files():
    # Мы добавляем все записи из каталога, которые
    являются # фактическими файлами и не начи-
    наются с "." global  serve_path
    return [file for file in os.listdir(serve_path) if os.path.isfile(os.path.join(serve_
↪path, file)) and file[:1] != "."]

# Когда клиент устанавливает соединение с нашим це-
левым сервером, эта функция будет вызвана со ссылкой
на это соединение. Затем мы отправляем клиенту #
список файлов, размещенных на сервере.
def client_connected(link):
    # Проверить, существует ли обслуживаемая директория
    if os.path.isdir(serve_path):
        RNS.log("Клиент подключен, отправка списка файлов...")

        link.set_link_closed_callback(client_disconnected)

        # Мы упаковываем список файлов для отправки в пакет
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        data = umsgpack.packb(list_files())

        # Проверить размер упакованных данных
        if len(data) <= RNS.Link.MDU:
            # Если он помещается в один пакет, мы просто #
            отправим его как один пакет по соединению.
            list_packet = RNS.Packet(link, data)
            list_receipt = list_packet.send()
            list_receipt.set_timeout(APP_TIMEOUT)
            list_receipt.set_delivery_callback(list_delivered)
            list_receipt.set_timeout_callback(list_timeout)
        else:
            RNS.log("Слишком много файлов в обслуживаемой директории!", RNS.LOG_ERROR)
            RNS.log("Вы должны реализовать функцию для разделения списка файлов на несколько
↪пакетов.", RNS.LOG_ERROR)
            RNS.log("Подсказка: Клиент уже поддерживает это :)", RNS.LOG_ERROR)

        # После этого мы'просто будем поддерживать соединение
        # открытым, пока клиент не запросит файл. Мы ' ll #
        настроим функцию, которая вызывается, когда # кли-
        ент отправляет пакет с запросом файла.
        link.set_packet_callback(client_request)
    else:
        RNS.log("Клиент подключен, но обслуживаемый путь больше не существует!", RNS.LOG_ERROR)
        link.teardown()

def client_disconnected(link):
    RNS.log("Клиент отключен")

def client_request(message, packet):
    global serve_path

    try:
        filename = message.decode("utf-8")
    except Exception as e:
        filename = None

    if filename in list_files():
        try:
            # Если у нас есть запрошенный файл, мы ' ll #
             прочитаем его и упакуем как ресурс
            RNS.log("Клиент запросил  \" "+filename+" \" ") file =
            open(os.path.join(serve_path, filename), "rb")

            file_resource = RNS.Resource(
                file,
                packet.link,
                callback=resource_sending_concluded
            )

            file_resource.filename = filename
        except Exception as e:
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            # Если что-то пошло не так, мы закрываем
            # ссылку
            RNS.log("Error while reading file  \""+filename+"\"", RNS.LOG_ERROR)
            packet.link.teardown()
            raise e
    else:
        # Если у нас его нет, мы закрываем ссылку
        RNS.log("Client requested an unknown file")
        packet.link.teardown()

# Эта функция вызывается на сервере, когда # пе-
редача ресурса завершается.
def resource_sending_concluded(resource):
    if hasattr(resource, "filename"):
        name = resource.filename
    else:
        name = "resource"

    if resource.status == RNS.Resource.COMPLETE:
        RNS.log("Done sending  \""+name+"\" to client")
    elif resource.status == RNS.Resource.FAILED:
        RNS.log("Sending \""+name+"\" to client failed")

def list_delivered(receipt):
    RNS.log("Список файлов был получен клиентом")

def list_timeout(receipt):
    RNS.log("Отправка списка клиенту превысила время ожидания, закрываем это соединение") link
    = receipt.destination link.
    teardown()

############################################################
#### Часть клиента #########################################
############################################################

# Мы храним глобальный список файлов, доступных на сервере
server_files       = []

# Ссылка на соединение с сервером
server_link        = None

# И ссылка на текущую загрузку
current_download   = None
current_filename   = None

# Переменные для хранения статистики
загрузки download_started = 0
download_finished = 0 download_time
= 0 transfer_size = 0 file_size

                  = 0
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# Эта инициализация выполняется, когда пользователь вы-
бирает # запуск в качестве клиента
def client(destination_hexhash, configpath):
    # Нам необходимо бинарное представление хеша назна-
    чения, # который был введен в командной строке
    try:
        dest_len = (RNS.Reticulum.TRUNCATED_HASHLENGTH//8)*2
        if len(destination_hexhash) != dest_len:
            raise ValueError(
                "Длина назначения недействительна, должно быть {hex} шестнадцатеричных символов (
↪{byte} байта).".format(hex=dest_len, byte=dest_len//2)
            )

        destination_hash = bytes.fromhex(destination_hexhash)
    except:
        RNS.log("Invalid destination entered. Check your input! \n")
        sys.exit(0)

    # Сначала мы должны инициализировать Reticulum
    reticulum = RNS.Reticulum(configpath)


    # Проверить, известен ли путь к назначению
    if not RNS.Transport.has_path(destination_hash):
        RNS.log("Destination is not yet known. Requesting path and waiting for announce␣
↪to arrive...")
        RNS.Transport.request_path(destination_hash)
        while not RNS.Transport.has_path(destination_hash):
            time.sleep(0.1)

    # Запоминаем идентификатор сервера
    server_identity = RNS.Identity.recall(destination_hash)

    # Информируем пользователя о начале подключения
    RNS.log("Установление связи с сервером...")

    # Когда идентификатор сервера известен, мы устанавливаем
    # назначение
    server_destination = RNS.Destination(
        server_identity,
        RNS.Destination.OUT,
        RNS.Destination.SINGLE,
        APP_NAME,
        "filetransfer",
        "server"
    )

    # Мы также хотим автоматически подтверждать входящие пакеты
    server_destination.set_proof_strategy(RNS.Destination.PROVE_ALL)

    # И создаем ссылку
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    link = RNS.Link(server_destination)

    # Мы ожидаем, что любые обычные пакеты данных по каналу
    # будут содержать список обслуживаемых файлов, поэтому мы устанавливаем
    # соответствующий коллбэк
    link.set_packet_callback(filelist_received)

    # Мы также настроим функции, чтобы информировать
    # пользователя, когда ссылка установлена или закрыта
    link.set_link_established_callback(link_established)
    link.set_link_closed_callback(link_closed)

    # И устанавливаем, чтобы канал автоматически начинал
    # загрузку анонсированных ресурсов
    link.set_resource_strategy(RNS.Link.ACCEPT_ALL)
    link.set_resource_started_callback(download_began)
    link.set_resource_concluded_callback(download_concluded)

    menu()

# Запрашивает указанный файл у сервера
def download(filename):
    global server_link, menu_mode, current_filename, transfer_size, download_started
    current_filename = filename
    download_started = 0
    transfer_size    = 0

    # Мы просто создаем пакет, содержащий запро-
    шенное имя файла, и отправляем его по каналу.
     Мы также указываем, что нам не требуется
    квитанция о получении пакета.
    request_packet = RNS.Packet(server_link, filename.encode("utf-8"), create_
↪receipt=False)
    request_packet.send()

    print("")
    print(("Запрошено  \" "+filename+" \" с сервера, ожидание начала загрузки..."))menu
    _mode = "download_started"

# Эта функция запускает простое меню для пользователя
, чтобы выбрать файлы для загрузки или выйти
menu_mode = None
def menu():
    global server_files, server_link
    # Ждем, пока у нас не будет списка файлов
    while len(server_files) == 0:
        time.sleep(0.1)
    RNS.log("Готово!")
    time.sleep(0.5)

    global menu_mode
    menu_mode = "main"
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    should_quit = False
    while (not should_quit):
        print_menu()

        while not menu_mode == "main":
            # Ждем
            time.sleep(0.25)

        user_input = input()
        if user_input == "q" or user_input == "quit" or user_input == "exit":
            should_quit = True
            print("")
        else:
            if user_input in server_files:
                download(user_input)
            else:
                try:
                    if 0 <= int(user_input) < len(server_files):
                        download(server_files[int(user_input)])
                except:
                    pass

    if should_quit:
        server_link.teardown()

# Выводит меню или экраны для различных
состояний клиентской программы.
# Это'просто и довольно неинтересно.
# Я не буду'вдаваться в подробности здесь. Просто
# строки в основном.
def print_menu():
    global menu_mode, download_time, download_started, download_finished, transfer_size,␣
↪file_size

    if menu_mode == "main":
        clear_screen()
        print_filelist()
        print("")
        print("Выберите файл для загрузки, введя имя или номер, или q для выхода") print(
        ("> "), end= ' ' ) elif menu
    _mode == "download_started":
        download_began = time.time()
        while menu_mode == "download_started":
            time.sleep(0.1)
            if time.time() > download_began+APP_TIMEOUT:
                print("Загрузка прервана по таймауту")
                time.sleep(1)
                server_link.teardown()

    if menu_mode == "downloading":
        print("Загрузка началась")
        print("")
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        while menu_mode == "downloading":
            global current_download
            percent = round(current_download.get_progress() * 100.0, 1)
            print((" \r Прогресс: "+str(percent)+" % "), end= ' ' ) sys
            .stdout.flush() time.sleep(0.1)


    if menu_mode == "save_error":
        print((" \r Прогресс: 100.0 %"), end= ' ' )
         sys.stdout.flush() print("")

        print("Не удалось записать загруженный файл на
        диск") current_download.status = RNS.Resource.
        FAILED menu_mode = "download_concluded"

    if menu_mode == "download_concluded":
        if current_download.status == RNS.Resource.COMPLETE:
            print(("\rProgress: 100.0 %"), end= ' ')
            sys.stdout.flush()

            # Печать статистики
            hours, rem = divmod(download_time, 3600)
            minutes, seconds = divmod(rem, 60)
            timestring = " {:0>2}:{:0>2}:{:05.2f}".format(int(hours),int(minutes),seconds)
            print("")
            print("")
            print("--- Статистика -----")
            print("\tВремя затрачено   : "+timestring)
            print("\tРазмер файла      : "+size_str(file_size)) print("
            \t Передано данных : "+size_str(transfer_size))
            print("\tЭффективная скорость : "+size_str(file_size/download_time, suffix='b')+
↪"/s")
            print("\tСкорость передачи : "+size_str(transfer_size/download_time, suffix='b
↪')+"/s")
            print("")
            print("Загрузка завершена! Нажмите Enter, чтобы вернуться в меню.")
            print("")
            input()

        else:
            print("")
            print("Загрузка не удалась! Нажмите Enter, чтобы вернуться в
            меню.") input()

        current_download =  None
        menu_mode = "main"
        print_menu()

# Эта функция выводит список файлов
# на подключенном сервере.
def print_filelist():
    global server_files
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    print("Файлы на сервере:")
    for index,file in enumerate(server_files):
        print("\t("+str(index)+")\t"+file)

def filelist_received(filelist_data, packet):
    global server_files, menu_mode
    try:
        # Распакуйте список и расширьте наш
        # локальный список доступных файлов
        filelist = umsgpack.unpackb(filelist_data)
        for file in filelist:
            if not file in server_files:
                server_files.append(file)

        # Если меню уже отображается,
        # мы обновим его полученными данными
        # только что получено
        if menu_mode == "main":
            print_menu()
    except:
        RNS.log("Получены неверные данные списка файлов, закрытие соединения") packet
        .link.teardown()

# Эта функция вызывается, когда связь #
 была установлена с сервером
def link_established(link):
    # Мы сохраняем ссылку на экземпляр # связи
    для дальнейшего использования
    global server_link
    server_link = link

    # Информируем пользователя о том,
    что сервер # подключен
    RNS.log("Соединение с сервером установлено")
    RNS.log("Ожидание списка файлов...")

    # И настроим небольшую задачу для проверки
    # потенциального таймаута при получении
    # списка файлов
    thread = threading.Thread(target=filelist_timeout_job, daemon= True)
    thread.start()

# Эта задача просто ожидает указанное #
время, а затем проверяет, был ли список
файлов # получен. Если нет, программа #
завершит работу.
def filelist_timeout_job():
    time.sleep(APP_TIMEOUT)

    global server_files
    if len(server_files) == 0:
        RNS.log("Истек таймаут ожидания списка файлов, выход")
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        sys.exit(0)


# Когда связь закрыта , мы'll информируем
# пользователя и выходим из программы
def link_closed(link):
    if link.teardown_reason == RNS.Link.TIMEOUT:
        RNS.log("The link timed out, exiting now")
    elif link.teardown_reason == RNS.Link.DESTINATION_CLOSED:
        RNS.log("The link was closed by the server, exiting now")
    else:
        RNS.log("Link closed, exiting now")

    time.sleep(1.5)
    sys.exit(0)

# Когда RNS обнаруживает, что загрузка
# началась , мы обновим состояние нашего меню
# чтобы пользователю был показан прогресс
# загрузки .
def download_began(resource):
    global menu_mode, current_download, download_started, transfer_size, file_size
    current_download = resource

    if download_started == 0:
        download_started = time.time()

    transfer_size += resource.size
    file_size = resource.total_size

    menu_mode = "downloading"

# Когда загрузка завершится , успешно # или нет , мы '
ll обновим состояние нашего меню и # проинформируем
пользователя о том , как все прошло .
def download_concluded(resource):
    global menu_mode, current_filename, download_started, download_finished, download_
↪time
    download_finished = time.time()
    download_time = download_finished - download_started

    saved_filename = current_filename

    if resource.status == RNS.Resource.COMPLETE:
        counter = 0
        while os.path.isfile(saved_filename):
            counter += 1
            saved_filename = current_filename+"."+str(counter)

        try:
            file = open(saved_filename, "wb")
            file.write(resource.data.read())
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            file.close()
            menu_mode = "download_concluded"
        except:
            menu_mode = "save_error"
    else:
        menu_mode = "download_concluded"

# Вспомогательная функция для вывода удобочи-
таемого размера файла
def size_str(num, suffix='B'):
    units = ['','Ki','Mi','Gi','Ti','Pi','Ei','Zi']
    last_unit = 'Yi'

    if suffix == 'b':
        num *= 8
        units = ['','K','M','G','T','P','E','Z']
        last_unit = 'Y'

    for unit in units:
        if abs(num) < 1024.0:
            return "%3.2f %s%s" % (num, unit, suffix)
        num /= 1024.0
    return "%.2f %s%s" % (num, last_unit, suffix)

# Удобная функция для очистки экрана
def clear_screen():
    os.system('cls' if os.name=='nt' else 'clear')

############################################################
#### Запуск программы ######################################
############################################################

# Эта часть программы запускается при старте,
# и анализирует ввод пользователя, а затем
# запускает нужный режим программы.
if __name__ == "__main__":
    try:
        parser = argparse.ArgumentParser(
            description="Простая утилита сервера и клиента для передачи файлов"
        )

        parser.add_argument(
            "-s",
            "--serve",
            action="store",
            metavar="dir",
            help="предоставлять каталог файлов клиентам"
        )

        parser.add_argument(
            "--config",
            action="store",
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            default=None,
            help="путь к альтернативному каталогу конфигурации Reticulum",
            type=str
        )

        parser.add_argument(
            "destination",
            nargs="?",
            default=None,
            help="шестнадцатеричный хеш назначения сервера",
            type=str
        )

        args = parser.parse_args()

        if args.config:
            configarg = args.config
        else:
            configarg = None

        if args.serve:
            if os.path.isdir(args.serve):
                server(configarg, args.serve)
            else:
                RNS.log("The specified directory does not exist")
        else:
            if (args.destination == None):
                print("")
                parser.print_help()
                print("")
            else:
                client(args.destination, configarg)

    except KeyboardInterrupt:
        print("")
        sys.exit(0)
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/Filetransfer.py.

9.11 Пользовательские интерфейсы

Пример *ExampleInterface* демонстрирует создание пользовательских интерфейсов для Reticulum. Reticulum может загружать и использовать любое количество пользовательских интерфейсов, которые будут полностью эквивалентны встроенным интерфейсам, включая все поддерживаемые режимы интерфейса и *общие параметры конфигурации* .

```
# Этот пример иллюстрирует создание пользовательского определения интерфейса #, которое может быть загружено и использовано Reticulum во
время выполнения #. Может быть создано # и загружено любое количество
пользовательских интерфейсов. Чтобы использовать интерфейс, поместите его в папку # ~/.reticulum/interfaces и добавьте запись интерфейса
в # ваш файл конфигурации Reticulum, подобную этой:
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
#  [[Пример пользовательского интерфейса]]
#    type = ExampleInterface
#    enabled = no
#    mode = gateway
#    port = /dev/ttyUSB0
#    speed = 115200
#    databits = 8
#    parity = none
#    stopbits = 1

from time import  sleep
import sys
import threading
import time

# Этот вспомогательный класс HDLC используется интерфейсом
# для разграничения и пакетирования данных по физическому
# носителю — в данном случае, последовательному соединению.
class HDLC():
    # Этот пример интерфейса пакетирует данные, используя
    # упрощенное кадрирование HDLC, аналогичное PPP
    FLAG      = 0x7E
    ESC       = 0x7D
    ESC_MASK = 0x20

    @staticmethod
    def escape(data):
        data = data.replace(bytes([HDLC.ESC]), bytes([HDLC.ESC, HDLC.ESC^HDLC.ESC_MASK]))
        data = data.replace(bytes([HDLC.FLAG]), bytes([HDLC.ESC, HDLC.FLAG^HDLC.ESC_
↪MASK]))
        return data

# Давайте ' s определим наш пользовательский класс интерфейса
. Он должен # быть подклассом класса RNS "Interface".
class ExampleInterface(Interface):
    # Все классы интерфейсов должны определять размер #
    IFAC по умолчанию, используемый при настройке IFAC,
    если пользователь # не указал пользовательский размер
    IFAC. Эта # опция задается в байтах.
    DEFAULT_IFAC_SIZE = 8

    # Следующие свойства являются локальными для этой
    # конкретной реализации интерфейса.
    owner     = None
    port      = None
    speed     = None
    databits =  None
    parity    = None
    stopbits =  None
    serial    = None

    # Все интерфейсы Reticulum должны иметь метод __init__
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    # который принимает 2 позиционных аргумента:
    # Экземпляр транспортного протокола RNS владельца и словарь
    # значений конфигурации.
    def __init__(self, owner, configuration):

        # Следующие строки демонстрируют обработку
        # потенциальных зависимостей, необходимых для
        # корректной работы интерфейса.
        import importlib
        if importlib.util.find_spec('serial') != None:
            import serial
        else:
            RNS.log("Использование этого интерфейса требует установки модуля последовательной связи. ", RNS.LOG_CRITICAL)
↪установлен.", RNS.LOG_CRITICAL)
            RNS.log("Вы можете установить его с помощью команды: python3 -m pip install␣
↪pyserial", RNS.LOG_CRITICAL)
            RNS.panic()

        # Начинаем с инициализации суперкласса
        super().__init__()

        # Чтобы убедиться, что данные конфигурации находятся в
        # правильном формате, мы разбираем их с помощью следу-
        ющего # метода в общем классе Interface. Этот шаг #
        необходим для обеспечения совместимости на всех # плат-
        формах, которые поддерживает Reticulum.
        ifconf    = Interface.get_config_obj(configuration)

        # Читаем имя интерфейса из конфигурации
        # и устанавливаем его для нашего экземпляра интерфейса.
        name      = ifconf["name"]
        self.name = name

        # Мы считываем параметры конфигурации из предоставлен-
        ных # данных конфигурации и предоставляем значения по
        умолчанию # в случае отсутствия каких-либо из них.
        порт      = ifconf["port"] if "port" in ifconf else None
        speed     = int(ifconf["speed"]) if "speed" in ifconf else 9600
        databits  = int(ifconf["databits"]) if "databits" in ifconf else 8
        parity    = ifconf["parity"] if "parity" in ifconf else "N"
        stopbits  = int(ifconf["stopbits"]) if "stopbits" in ifconf else 1

        # Если порт не указан, мы прерываем настройку, #
        вызывая исключение.
        if port == None:
            raise ValueError(f"Порт не указан для     {self}")

        # Все интерфейсы должны предоставлять значение MTU оборудования # Экземпляру транспорт-
        ного протокола RNS. Это значение должно # быть макси-
        мальным размером полезной нагрузки пакета данных, кото-
        рый # базовый носитель способен обрабатывать во всех #
        случаях без какой-либо сегментации.
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        self.HW_MTU = 564

        # Изначально мы устанавливаем свойство «online» в
        значение false, # поскольку интерфейс еще не был
        полностью # инициализирован и подключен.
        self.online = False

        # В этом случае мы также можем установить указанную битовую-
        # скорость интерфейса на скорость последовательного порта.
        self.bitrate  = speed

        # Настроить внутренние свойства интерфейса
        # в соответствии с предоставленной конфигурацией.
        self.pyserial = serial
        self.serial   = None
        self.owner    = owner
        self.port     = port
        self.speed    = speed
        self.databits = databits
        self.parity   = serial.PARITY_NONE
        self.stopbits = stopbits
        self.timeout  = 100

        if parity.lower() == "e"  or parity.lower() == "even":
            self.parity = serial.PARITY_EVEN

        if parity.lower() == "o"  or parity.lower() == "odd":
            self.parity = serial.PARITY_ODD

        # Поскольку все необходимые параметры теперь настроены,
        # мы попытаемся открыть последовательный порт.
        try:
            self.open_port()
        except Exception as e:
            RNS.log("Could not open serial port for interface "+str(self), RNS.LOG_ERROR)
            raise e

        # Если открытие порта удалось, запустите любую
        требуемую конфигурацию после открытия.
        if self.serial.is_open:
            self.configure_device()
        else:
            raise IOError("Could not open serial port")

    # Открыть последовательный порт с заданной конфигурацией
    # параметры и сохраняет ссылку на открытый порт.
    def open_port(self):
        RNS.log("Открытие последовательного порта "+self.port+"...", RNS.LOG_VERBOSE) self.
        serial = self.pyserial.Serial( port =
            self.port, baudrate
            = self.speed, bytesize =
            self.databits,
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            parity = self.parity,
            stopbits = self.stopbits,
            xonxoff = False,
            rtscts = False,
            timeout = 0,
            inter_byte_timeout = None,
            write_timeout = None,
            dsrdtr = False,
        )

    # Единственное, что требуется после открытия порта, #
    это подождать немного времени для инициализации # обо-
    рудования, а затем запустить поток, # который считыва-
    ет любые входящие данные с устройства.
    def configure_device(self):
        sleep(0.5)
        thread = threading.Thread(target=self.read_loop)
        thread.daemon = True
        thread.start()
        self.online = True
        RNS.log("Последовательный порт "+self.port+" теперь открыт", RNS.LOG_VERBOSE)


    # Этот метод будет вызываться из нашего цикла
    чтения # всякий раз, когда полный пакет был полу-
    чен через # базовую среду.
    def process_incoming(self, data):
        # Обновляем счетчик полученных байтов
        self.rxb += len(data)

        # И отправляем пакет данных в Экземпляр транспортного протокола
        # для обработки.
        self.owner.inbound(data, self)

    # Запущенный Экземпляр транспортного протокола
    Reticulum будет # вызывать этот метод на интерфейсе
    всякий раз, когда # интерфейс должен передать пакет.
    def process_outgoing(self,data):
        if self.online:
            # Сначала экранируем и пакетируем данные
            # в соответствии с кадрированием HDLC.
            data = bytes([HDLC.FLAG])+HDLC.escape(data)+bytes([HDLC.FLAG])

            # Затем записываем кадрированные данные в порт
            written = self.serial.write(data)

            # Обновляем счетчик переданных байтов
            # и убеждаемся, что все данные были записаны
            self.txb += len(data)
            if written != len(data):
                raise IOError("Последовательный интерфейс записал только "+str(written)+" байт из
↪"+str(len(data)))
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    # Этот цикл чтения выполняется в потоке и постоянно # при-
    нимает байты из базового последовательного порта.
    # Когда полный пакет будет получен, он будет # отправлен
    в метод process_incoming, который # в свою очередь пере-
    даст его экземпляру транспортного протокола.
    def read_loop(self):
        try:
            in_frame = False
            escape = False
            data_buffer = b""
            last_read_ms = int(time.time()*1000)

            while self.serial.is_open:
                if self.serial.in_waiting:
                    byte = ord(self.serial.read(1))
                    last_read_ms = int(time.time()*1000)

                    if (in_frame and byte == HDLC.FLAG):
                        in_frame = False
                        self.process_incoming(data_buffer)
                    elif (byte == HDLC.FLAG):
                        in_frame = True
                        data_buffer = b""
                    elif (in_frame and len(data_buffer) < self.HW_MTU):
                        if (byte == HDLC.ESC):
                            escape = True
                        else:
                            if (escape):
                                if (byte == HDLC.FLAG ^ HDLC.ESC_MASK):
                                    byte = HDLC.FLAG
                                if (byte == HDLC.ESC   ^ HDLC.ESC_MASK):
                                    byte = HDLC.ESC
                                escape = False
                            data_buffer = data_buffer+bytes([byte])

                else:
                    time_since_last = int(time.time()*1000) - last_read_ms
                    if len(data_buffer) > 0 and time_since_last > self.timeout:
                        data_buffer = b""
                        in_frame = False
                        escape = False
                    sleep(0.08)

        except Exception as e:
            self.online = False
            RNS.log("Произошла ошибка последовательного порта, содержавшееся исключение: "+str(e),
↪ RNS.LOG_ERROR)
            RNS.log("Интерфейс "+str(self)+" столкнулся с невосстановимой ошибкой и␣
↪теперь в оффлайне.", RNS.LOG_ERROR)

            if RNS.Reticulum.panic_on_interface_error:
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
                RNS.panic()

            RNS.log("Reticulum будет пытаться периодически переподключать интерфейс.",
↪RNS.LOG_ERROR)

        self.online = False
        self.serial.close()
        self.reconnect_port()

    # Этот метод обрабатывает отключения последовательного порта.
    def reconnect_port(self):
        while not self.online:
            try:
                time.sleep(5)
                RNS.log("Попытка переподключить последовательный порт "+str(self.port)+" для
↪"+str(self)+"...", RNS.LOG_VERBOSE)
                self.open_port()
                if self.serial.is_open:
                    self.configure_device()
            except Exception as e:
                RNS.log("Ошибка при переподключении порта, содержащееся исключение было:
↪"+str(e), RNS.LOG_ERROR)

        RNS.log("Последовательный порт для "+str(self)+" переподключен")

    # Сообщить Reticulum, что этот интерфейс не должен вы-
    полнять никаких ограничений на входящий трафик.
    def should_ingress_limit(self):
        return False

    # Мы должны предоставить строковое представление этого #
    интерфейса, которое используется всякий раз, когда интерфейс
    # выводится в журналах или внешних программах.
    def __str__(self):
        return "ExampleInterface["+self.name+"]"

# Наконец, зарегистрируйте определённый класс интерфейса как
# целевой класс, который Reticulum будет использовать в качестве интерфейса
interface_class = ExampleInterface
```

Этот пример также можно найти по адресу https://github.com/markqvist/Reticulum/blob/master/Examples/ExampleInterface.py.

ГЛАВА

ДЕСЯТЬ

СПРАВОЧНИК ПО API

Связь по сетям Reticulum достигается с использованием простого набора классов, предоставляемых API RNS. Эта глава перечисляет и объясняет все классы, предоставляемые API Сетевого стека Reticulum, а также их сигнатуры методов и использование. Его можно использовать в качестве справочника при написании приложений, использующих Reticulum, или его можно прочитать полностью, чтобы получить полное представление о функциональности RNS с точки зрения разработчика.

10.1 Reticulum

class RNS.**Reticulum**(*configdir=None*, *loglevel=None*, *logdest=None*, *verbosity=None*, *require_shared_instance=False*, *shared_instance_type=None*)

Этот класс используется для инициализации доступа к Reticulum в рамках программы. Необходимо создать ровно один экземпляр этого класса перед выполнением любых других операций RNS, таких как создание назначений или отправка трафика. Каждая независимо выполняемая программа должна создать свой собственный экземпляр класса Reticulum, но Reticulum автоматически обрабатывает межпрограммную связь в рамках одной системы и предоставляет все подключенные программы внешним интерфейсам.

Как только экземпляр этого класса создан, Reticulum начинает открывать и конфигурировать любые аппаратные устройства, указанные в предоставленной конфигурации.

В настоящее время первый запущенный экземпляр должен оставаться активным, пока подключены другие локальные экземпляры , поскольку первый созданный экземпляр будет действовать как главный, напрямую обменивающийся данными с внешним оборудованием, таким как модемы, TNC и радиостанции. Если главному экземпляру предписано завершить работу, он не завершится до тех пор, пока все клиентские процессы не будут прекращены (если только они не были принудительно остановлены).

Если вы запускаете Reticulum на системе, где несколько различных программ используют RNS, запускаясь и завершаясь в разное время, будет выгодно запустить главный экземпляр RNS в качестве демона, чтобы другие программы могли использовать его по запросу.

MTU = 500

MTU, которому Reticulum соответствует и ожидает, что другие узлы будут ему соответствовать. По умолчанию MTU составляет 500 байт. В пользовательских реализациях сети RNS возможно изменить это значение, но это полностью нарушит совместимость со всеми другими сетями RNS. Идентичный MTU является предварительным условием для обмена данными между узлами в одной сети.

Если вы действительно не знаете, что делаете, MTU следует оставить значением по умолчанию.

LINK_MTU_DISCOVERY = True

Включено ли автоматическое обнаружение MTU канала по умолчанию в этом выпуске. Обнаружение MTU канала значительно увеличивает пропускную способность по быстрым каналам, но требует, чтобы все промежуточные узлы также поддерживали его. Поддержка этой функции была добавлена в RNS версии 0.9.0. Эта опция будет включена по умолчанию в ближайшем будущем . Пожалуйста, обновите свои экземпляры RNS.

ANNOUNCE_CAP = 2

Максимальный процент пропускной способности интерфейса, который в любой момент времени может быть использован для распространения объявлений. Если объявление было запланировано для трансляции на интерфейсе, но это привело бы к превышению выделенной пропускной способности, объявление будет поставлено в очередь на передачу, когда пропускная способность станет доступной.

Reticulum всегда будет отдавать приоритет распространению объявлений с меньшим количеством переходов, гарантируя, что удаленные, крупные сети со множеством пиров на быстрых каналах не перегружают пропускную способность меньших сетей на более медленных средах. Если объявление остается в очереди в течение длительного времени, оно в конечном итоге будет отброшено.

Это значение будет применяться по умолчанию ко всем созданным интерфейсам, но его можно настроить индивидуально для каждого интерфейса. В общем случае, глобальная настройка по умолчанию не должна изменяться, и любые корректировки следует производить на основе каждого интерфейса.

MINIMUM_BITRATE = 5

Минимальная скорость передачи данных, требуемая для среды, чтобы Reticulum мог успешно устанавливать соединения. В настоящее время 5 бит в секунду.

static get_instance()

Возвращает текущий запущенный экземпляр Reticulum.

static should_use_implicit_proof()

Возвращает, являются ли отправленные подтверждения явными или неявными.

Возвращает

True, если текущая запущенная конфигурация указывает использовать неявные подтверждения. False в противном случае.

static transport_enabled()

Возвращает, включён ли Transport для запущенного экземпляра.

Когда Transport включён, Reticulum будет маршрутизировать трафик для других узлов, отвечать на запросы путей и передавать объявления по сети.

Возвращает

True, если Transport включён, False в противном случае.

static link_mtu_discovery()

Возвращает, включено ли обнаружение MTU канала для запущенного экземпляра.

Когда обнаружение MTU канала включено, Reticulum будет автоматически обновлять MTU канала до максимально поддерживаемого значения, увеличивая скорость и эффективность передачи данных.

Возвращает

True, если обнаружение MTU канала включено, False в противном случае.

static remote_management_enabled()

Возвращает, включено ли удаленное управление для запущенного экземпляра.

Когда удаленное управление включено, аутентифицированные узлы могут удаленно запрашивать данные и управлять этим экземпляром.

Возвращает

True, если удалённое управление включено; False, если нет.

10.2 Идентичность

class RNS.Identity(*create_keys=True*)

Этот класс используется для управления идентификаторами в Reticulum. Он предоставляет методы для шифрования, дешифрования, подписей и проверки, являясь основой для всей зашифрованной связи в сетях Reticulum.

Параметры
create_keys – Указывает, должны ли быть сгенерированы новые ключи шифрования и подписи.

CURVE = 'Curve25519'

Кривая, используемая для обмена ключами Диффи-Хеллмана на эллиптических кривых.

KEYSIZE = 512

Размер ключа X.25519 в битах. Полный ключ представляет собой конкатенацию 256-битного ключа шифрования и 256-битного ключа подписи.

RATCHETSIZE = 256

Размер ключа храповика X.25519 в битах.

RATCHET_EXPIRY = 2592000

Срок действия полученных ключей храповика в секундах, по умолчанию 30 дней. Reticulum всегда будет использовать самый последний объявленный храповик и запоминать его в течение до RATCHET_EXPIRY с момента получения, после чего он будет отброшен. Если тем временем будет объявлен новый ретчет, он заменит уже известный ретчет .

TRUNCATED_HASHLENGTH = 128

Константа, указывающая усеченную длину хеша (в битах), используемую Reticulum для адресуемых хешей и других целей. Не настраивается.

static recall(*target_hash*, *from_identity_hash=False*)

Вызов идентификатора для хеша назначения или идентификатора. По умолчанию эта функция вернет идентификатор, связанный с заданным хешем *назначения* . Например, если вы знаете хеш назначения lxmf.delivery конечной точки, эта функция вернет соответствующий базовый идентификатор. Вы также можете выполнить поиск идентификатора по известному *хешу идентификатора* , установив аргумент from_identity_hash .

Параметры

- **target_hash** – Хеш назначения или идентификатора в виде *bytes*.
- **from_identity_hash** – Искать ли по хешу идентификатора вместо хеша назначения в виде *bool* .

Возвращает
Экземпляр *RNS.Identity* , который может быть использован для создания исходящего *RNS.Destination* , или *None* , если назначение неизвестно.

static recall_app_data(*destination_hash*)

Вызывает последние полученные app_data для хеша назначения.

Параметры
destination_hash – Хеш назначения в формате *bytes*.

Возвращает
Bytes содержащие app_data, или *None*, если назначение неизвестно.

static full_hash(*data*)

Получить хеш SHA-256 переданных данных.

Параметры
data – Данные для хеширования в формате *bytes*.

Возвращает
хеш SHA-256 в формате *bytes*.

static truncated_hash(*data*)

Получить усеченный хеш SHA-256 переданных данных.

Параметры
data – Данные для хеширования в формате *bytes*.

Возвращает
Усеченный хеш SHA-256 в формате *bytes*.

static get_random_hash()

Получить случайный хеш SHA-256.

Параметры
data – Данные для хеширования в формате *bytes*.

Возвращает
Усеченный хеш SHA-256 случайных данных в формате *bytes*.

static current_ratchet_id(*destination_hash*)

Получить ID используемого в данный момент ключа ratchet для заданного хеша назначения

Параметры
destination_hash – Хеш назначения в формате *bytes*.

Возвращает
Идентификатор ratchet в виде *bytes*или *None*.

static from_bytes(*prv_bytes*)

Создает новый экземпляр *RNS.Identity* из *bytes* приватного ключа. Может использоваться для загрузки ранее созданных и сохраненных идентификаторов в Reticulum.

Параметры
prv_bytes – *bytes* сохраненного приватного ключа. ОПАСНО! Никогда не используйте это для генерации нового ключа путем передачи случайных данных в prv_bytes.

Возвращает
Экземпляр*RNS.Identity*или *None*, если данные *bytes*были недействительны.

static from_file(*path*)

Создает новый экземпляр *RNS.Identity* из файла. Может использоваться для загрузки ранее созданных и сохраненных идентификаторов в Reticulum.

Параметры
path – Полный путь к сохраненным данным *RNS.Identity*

Возвращает
Экземпляр*RNS.Identity*или *None*, если загруженные данные были недействительны.

to_file(*path*)

Сохраняет идентификатор в файл. Это запишет закрытый ключ на диск, и любой, кто имеет доступ к этому файлу, сможет расшифровать всю связь для данного идентификатора. Будьте очень осторожны с этим методом.

Параметры
path – Полный путь, указывающий, где следует сохранить идентификатор.

Возвращает
True, если файл был сохранён, иначе False.

get_private_key()

Возвращает
Приватный ключ в виде *bytes*

`get_public_key`()

Возвращает
Публичный ключ в виде *bytes*

`load_private_key`(*prv_bytes*)
Загрузить приватный ключ в экземпляр.

Параметры
prv_bytes – Приватный ключ в виде *bytes*.

Возвращает
True, если ключ был загружен, иначе False.

`load_public_key`(*pub_bytes*)
Загрузить публичный ключ в экземпляр.

Параметры
pub_bytes – Публичный ключ в виде *bytes*.

Возвращает
True, если ключ был загружен, иначе False.

`encrypt`(*plaintext*, *ratchet=None*)
Шифрует информацию для идентификатора.

Параметры
plaintext – Открытый текст для шифрования в виде *bytes*.

Возвращает
Токен шифротекста в виде *bytes*.

Вызывает
KeyError, если экземпляр не содержит открытого ключа.

`decrypt`(*ciphertext_token*, *ratchets=None*, *enforce_ratchets=False*, *ratchet_id_receiver=None*)
Расшифровывает информацию для идентификатора.

Параметры
ciphertext – Зашифрованный текст для расшифровки в виде *bytes*.

Возвращает
Открытый текст в виде *bytes*, или *None*, если расшифровка не удалась.

Вызывает
KeyError, если экземпляр не содержит закрытого ключа.

`sign`(*message*)
Подписывает информацию идентификатором.

Параметры
message – Сообщение для подписи в виде *bytes*.

Возвращает
Подпись в виде *bytes*.

Вызывает
KeyError, если экземпляр не содержит закрытого ключа.

`validate`(*signature*, *message*)
Проверяет подпись подписанного сообщения.

Параметры

- **signature** – Подпись для проверки в виде *bytes*.
- **message** – Сообщение для проверки в виде *bytes*.

Возвращает
True, если подпись действительна; в противном случае False.

Вызывает
KeyError, если экземпляр не содержит открытого ключа.

10.3 Назначение

class RNS.**Destination**(*identity*, *direction*, *type*, *app_name*, **aspects*)

Класс, используемый для описания конечных точек в сети Reticulum. Экземпляры Destination используются как для создания исходящих, так и для входящих конечных точек. Тип назначения будет определять, используется ли шифрование и какой тип шифрования используется при обмене данными с конечной точкой. Адресат также может объявить о своем присутствии в сети, что позволит распространить необходимые ключи для зашифрованного обмена данными с ним.

Параметры

- **identity** – Экземпляр *RNS.Identity* . Может содержать только открытые ключи для исходящего адресата или закрытые ключи для входящего.
- **direction** – RNS.Destination.IN или RNS.Destination.OUT.
- **type** – RNS.Destination.SINGLE , RNS.Destination.GROUP или RNS.Destination.PLAIN .
- **app_name** – Строка, указывающая имя приложения.
- ***aspects** – Любое ненулевое количество строковых аргументов.

RATCHET_COUNT = 512

Количество сгенерированных ключей ratchet по умолчанию, которое будет сохранять адресат, если для него включены ratchets.

RATCHET_INTERVAL = 1800

Минимальный интервал между ротацией ключей храповика, в секундах.

static expand_name (*identity* , *app_name* , **aspects*)

Возвращает
Строка, содержащая полное удобочитаемое имя назначения для app_name и ряда аспектов.

static app_and_aspects_from_name (*full_name*)

Возвращает
Кортеж, содержащий имя приложения и список аспектов, для строки full-name.

static hash_from_name_and_identity (*full_name*, *identity*)

Возвращает
Имя назначения в форме адресного хеша, для полной строки имени и экземпляра Identity.

static hash (*identity*, *app_name*, **aspects*)

Возвращает
Имя назначения в форме адресного хеша, для app_name и ряда аспектов.

announce(*app_data=None* , *path_response=False* , *attached_interface=None* , *tag=None* , *send=True*) Создает пакет объявления для данного назначения и широковещательно рассылает его по всем соответствующим интерфейсам. Данные, специфичные для приложения, могут быть добавлены к объявлению.

Параметры

- **app_data** – *bytes* , содержащие app_data.
- **path_response** – Внутренний флаг, используемый *RNS.Transport*. Игнорировать.

accepts_links(*accepts=None*)

Устанавливает или запрашивает, принимает ли получатель входящие запросы на установление соединения.

Параметры

accepts – Если `True` или `False` , этот метод устанавливает, принимает ли получатель входящие запросы на установление соединения. Если не предоставлено или `None` , метод возвращает, принимает ли получатель в данный момент запросы на установление соединения.

Возвращает

`True` или `False` в зависимости от того, принимает ли получатель входящие запросы на установление соединения, если параметр *accepts* не предоставлен или `None` .

set_link_established_callback(*callback*)

Регистрирует функцию, которая будет вызвана при установлении соединения с этим получателем.

Параметры

callback – Функция или метод с сигнатурой *callback(link)* , который будет вызван при установлении нового соединения с этим получателем.

set_packet_callback(*callback*)

Регистрирует функцию, которая будет вызвана при получении пакета этим получателем.

Параметры

callback – Функция или метод с сигнатурой *callback(data, packet)* , которые вызываются при получении пакета этим пунктом назначения.

set_proof_requested_callback(*callback*)

Регистрирует функцию, которая будет вызываться при запросе подтверждения для пакета, отправленного в этот пункт назначения. Позволяет контролировать, когда и следует ли возвращать подтверждения для полученных пакетов.

Параметры

callback – Функция или метод с сигнатурой *callback(packet)* , которые будут вызываться при получении пакета, запрашивающего подтверждение. Обратный вызов должен возвращать True или False. Если обратный вызов возвращает True, подтверждение будет отправлено. Если он возвращает False, подтверждение не будет отправлено.

set_proof_strategy(*proof_strategy*)

Устанавливает стратегию подтверждения пункта назначения.

Параметры

proof_strategy – Один из `RNS.Destination.PROVE_NONE` , `RNS.Destination. PROVE_ALL` или `RNS.Destination.PROVE_APP` . Если установлено `RNS.Destination.PROVE_APP` , будет вызван *proof_requested_callback* для определения того, следует ли отправлять подтверждение.

register_request_handler(*path*, *response_generator=None*, *allow=ALLOW_NONE* , *allowed_list=None*, *auto_compress=True*)

Регистрирует обработчик запросов.

Параметры

- **path** – Путь для регистрации обработчика запросов.

- **response_generator** – Вызываемая функция или метод с сигнатурой *re- sponse_generator(path , data, request_id, link_id, remote_identity, requested_at)* . Все, что возвращает эта функция, будет отправлено в качестве ответа запрашивающей стороне. Если функция возвращает None , ответ не будет отправлен.
- **allow** – Одно из значений: RNS.Destination.ALLOW_NONE , RNS.Destination.ALLOW_ALL или RNS. Destination.ALLOW_LIST . Если установлено RNS.Destination.ALLOW_LIST , обработчик запросов будет отвечать только на запросы идентифицированных пиров из предоставленного списка.
- **allowed_list** – Список хешей*bytes-like RNS.Identity* .
- **auto_compress** – Если True или False , определяет, следует ли выполнять автоматическое сжатие ответов. Если установлено целочисленное значение, ответы будут автоматически сжиматься только в том случае, если их размер в байтах не превышает это значение. Если параметры опущены, будут использоваться настройки сжатия по умолчанию.

Вызывает

ValueError — если какой-либо из предоставленных аргументов недействителен.

deregister_request_handler (*path*)

Отменяет регистрацию обработчика запросов.

Параметры

path — Путь обработчика запросов, который будет дерегистрирован.

Возвращает

True, если обработчик был дерегистрирован, иначе False.

enable_ratchets(*ratchets_path*)

Включает «храповики» (ratchets) для адресата. Когда «храповики» включены, Reticulum будет автоматически ротировать ключи , используемые для шифрования пакетов для этого адресата, и включать последний ключ «храповика» в объявления.

Включение «храповиков» для адресата обеспечит прямую секретность для пакетов, отправленных этому адресату, даже если они отправлены вне Link . Стандартная процедура установки Link в Reticulum уже выполняет свой собственный обмен эфемерными ключами для каждой установки соединения, что означает, что «храповики» не требуются для обеспечения прямой секретности для соединений.

Включение «храповиков» незначительно повлияет на размер объявления, добавляя 32 байта к каждому отправляемому объявлению.

Параметры

ratchets_path – Путь к файлу для хранения данных храповика.

Возвращает

True, если операция прошла успешно, иначе False.

enforce_ratchets()

Когда принудительное использование храповика включено, данный пункт назначения никогда не будет принимать пакеты, использующие его базовый ключ идентификации для шифрования, а будет принимать только пакеты, зашифрованные одним из сохраненных ключей храповика.

set_retained_ratchets(*retained_ratchets*)

Устанавливает количество ранее сгенерированных ключей храповика, которое этот пункт назначения будет сохранять и пытаться использовать при расшифровке входящих пакетов. По умолчанию равно Destination.RATCHET_COUNT .

Параметры

retained_ratchets – Количество сгенерированных храповиков для сохранения.

Возвращает

True, если операция прошла успешно; False, если нет.

set_ratchet_interval(*interval*)

Устанавливает минимальный интервал в секундах между ротациями ключей храповика. По умолчанию равно `Destination.RATCHET_INTERVAL`.

Параметры
interval – Минимальный интервал в секундах.

Возвращает
True, если операция прошла успешно; False, если нет.

create_keys()

Для пункта назначения типа`RNS.Destination.GROUP`создает новый симметричный ключ.

Вызывает
`TypeError` при вызове для несовместимого типа назначения.

get_private_key()

Для назначения типа`RNS.Destination.GROUP`возвращает симметричный закрытый ключ.

Вызывает
`TypeError` при вызове для несовместимого типа назначения.

load_private_key(*key*)

Для назначения типа`RNS.Destination.GROUP`загружает симметричный закрытый ключ.

Параметры
key – объект *bytes-like*, содержащий симметричный ключ.

Вызывает
`TypeError` при вызове для несовместимого типа назначения.

encrypt(*plaintext*)

Шифрует информацию для назначения типа`RNS.Destination.SINGLE`или`RNS.Destination.GROUP`.

Параметры
plaintext – объект *bytes-like*, содержащий открытый текст для шифрования.

Вызывает
`ValueError` если назначение не содержит необходимого ключа для шифрования.

decrypt(*ciphertext*)

Расшифровывает информацию для назначения типа`RNS.Destination.SINGLE`или`RNS.Destination.GROUP`.

Параметры
ciphertext – *Bytes*,содержащий зашифрованный текст для расшифровки.

Вызывает
`ValueError` если назначение не содержит необходимого ключа для расшифровки.

sign(*message*)

Подписывает информацию для назначения типа`RNS.Destination.SINGLE`.

Параметры
message – *Bytes*,содержащий сообщение для подписи.

Возвращает
Объект *bytes-like*, содержащий подпись сообщения, или *None*, если назначение не смогло подписать сообщение.

set_default_app_data(*app_data=None*)

Устанавливает app_data по умолчанию для адресата. Если установлено, app_data по умолчанию будет включено в каждое объявление, отправленное адресатом, если только другое app_data не указано в методе *announce* .

Параметры

app_data – Объект типа *bytes-like* , содержащий app_data по умолчанию, или *callable* , возвращающий объект типа *bytes-like*, содержащий app_data.

clear_default_app_data()

Очищает app_data по умолчанию, ранее установленное для адресата.

10.4 Пакет

class RNS.**Packet**(*destination*, *data*, *create_receipt=True*)

Класс Packet используется для создания экземпляров пакетов, которые могут быть отправлены по сети Reticulum. Пакеты будут автоматически зашифрованы, если они адресованы назначению RNS.Destination.SINGLE , назначению RNS.Destination.GROUP или *RNS.Link* .

Для назначений RNS.Destination.GROUP Reticulum будет использовать предварительно разделенный ключ, настроенный для данного назначения. Все пакеты, предназначенные для группы, шифруются одним и тем же ключом AES-256.

Для RNS.Destination.SINGLE назначений Reticulum будет использовать новый эфемерный ключ AES-256 для каждого пакета.

Для *RNS.Link*назначений Reticulum будет использовать эфемерные ключи для каждого соединения и предлагает**прямую секретность**.

Параметры

- **destination** – Экземпляр*RNS.Destination*, которому будет отправлен пакет.
- **data** – Полезная нагрузка данных, которая будет включена в пакет в виде *bytes*.
- **create_receipt** – Указывает, следует ли создавать *RNS.PacketReceipt* при создании экземпляра пакета.

ENCRYPTED_MDU = 383

Максимальный размер полезной нагрузки данных в одном зашифрованном пакете

PLAIN_MDU = 464

Максимальный размер полезной нагрузки данных в одном незашифрованном пакете

send()

Отправляет пакет.

Возвращает

Экземпляр *RNS.PacketReceipt* , если *create_receipt* был установлен в *True* при создании экземпляра пакета ; в противном случае возвращает *None* . Если пакет не удалось отправить, возвращается *False* .

resend()

Повторно отправляет пакет.

Возвращает

Экземпляр *RNS.PacketReceipt* , если *create_receipt* был установлен в *True* при создании экземпляра пакета ; в противном случае возвращает *None* . Если пакет не удалось отправить, возвращается *False* .

get_rssi()

Возвращает

Уровень мощности принятого сигнала физического уровня*Received Signal Strength Indication*, если доступен; в противном случае — None.

get_snr()

Возвращает

Соотношение сигнал-шум физического уровня*Signal-to-Noise Ratio*, если доступно; в противном случае — None.

get_q()

Возвращает

Качество канала физического уровня*Link Quality*, если доступно; в противном случае — None.

10.5 Получение пакета

class RNS.**PacketReceipt**

Класс PacketReceipt используется для получения уведомлений об экземплярах *RNS.Packet* , отправленных по сети. Экземпляры этого класса никогда не создаются вручную, но всегда возвращаются методом *send()* экземпляра *RNS*.Packet .

get_status()

Возвращает

Статус связанного экземпляра *RNS.Packet* . Может быть одним из RNS. PacketReceipt. SENT , RNS.PacketReceipt.DELIVERED , RNS.PacketReceipt.FAILED или RNS. PacketReceipt.CULLED

get_rtt()

Возвращает

Время приёма-передачи в секундах

set_timeout(*timeout*)

Устанавливает таймаут в секундах

Параметры

timeout – Таймаут в секундах.

set_delivery_callback(*callback*)

Устанавливает функцию, которая вызывается, если успешная доставка была подтверждена.

Параметры

callback – Вызываемый объект с сигнатурой callback(packet_receipt)

set_timeout_callback(*callback*)

Устанавливает функцию, которая вызывается, если время доставки истекло.

Параметры

callback – Вызываемый объект с сигнатурой callback(packet_receipt)

10.6. Ссылка

class RNS.**Link**(*destination*, *established_callback=None*, *closed_callback=None*)

Этот класс используется для установления связей с другими пирами и управления ими. Когда создается экземпляр ссылки, Reticulum попытается установить проверенное и зашифрованное соединение с указанным пунктом назначения.

Параметры

- **destination** – Экземпляр*RNS.Destination*, к которому необходимо установить ссылку.

- **established_callback** – Необязательная функция или метод с сигнатурой *callback(link)* , который будет вызван при установлении ссылки.
- **closed_callback** – Необязательная функция или метод с сигнатурой *callback (link)* , вызываемый при закрытии соединения.

CURVE = 'Curve25519'

Кривая, используемая для обмена ключами Диффи-Хеллмана на эллиптических кривых.

ESTABLISHMENT_TIMEOUT_PER_HOP = 6

Тайм-аут для установки соединения в секундах на один хоп до получателя.

KEEPALIVE_TIMEOUT_FACTOR = 4

Фактор тайм-аута RTT, используемый при расчёте тайм-аута соединения.

STALE_GRACE = 5

Льготный период в секундах, используемый при расчёте тайм-аута соединения.

KEEPALIVE = 360

Интервал по умолчанию для отправки пакетов keep-alive по установленным соединениям в секундах.

STALE_TIME = 720

Если в течение этого периода не получено трафика или пакетов keep-alive, соединение будет помечено как неактивное, и будет отправлен финальный пакет keep-alive. Если после этого в течение `RTT * KEEPALIVE_TIMEOUT_FACTOR + STALE_GRACE` не получено трафика или пакетов keep-alive, соединение считается истекшим по тайм-ауту и будет разорвано.

identify(*identity*)

Идентифицирует инициатора соединения для удалённого пира. Это может произойти только после установления соединения и осуществляется по зашифрованному каналу. Идентификация раскрывается только удаленному узлу, и анонимность инициатора таким образом сохраняется. Этот метод может быть использован для аутентификации.

Параметры

identity – Экземпляр RNS.Identity для идентификации.

request(*path*, *data=None*, *response_callback=None*, *failed_callback=None*, *progress_callback=None*, *timeout=None*)

Отправляет запрос удаленному узлу.

Параметры

- **path** – Путь запроса.
- **response_callback** – Необязательная функция или метод с сигнатурой *re- sponse_callback(request_receipt)* , вызываемый при получении ответа. Дополнительную информацию см. в разделе *Request Example*.
- **failed_callback** – Необязательная функция или метод с сигнатурой *failed _callback(request_receipt)* , вызываемый при сбое запроса. Дополнительную информацию см. в разделе *Request Example*.
- **progress_callback** – *Необязательная функция или метод с сигнатурой progress_callback(request_receipt)* , вызываемый по мере получения ответа. Доступ к прогрессу можно получить как число с плавающей запятой от 0.0 до 1.0 через свойство *re- quest_receipt.progress* .
- **timeout** – Необязательный тайм-аут запроса в секундах. Если передано *None* , оно будет рассчитано на основе RTT канала.

Возвращает

Экземпляр*RNS.RequestReceipt*, если запрос был отправлен, или *False*, если нет.

track_phy_stats(*track*)

Вы можете включить статистику физического уровня для каждого канала. Если эта функция включена и канал работает через интерфейс, поддерживающий отчетность о статистике физического уровня, вы сможете получить такие данные, как *RSSI* , *SNR* и физическое *Link Quality* для канала.

Параметры

track – Отслеживать или нет статистику физического уровня. Значение должно быть `True` или `False` .

`get_rssi` ()

Возвращает

Индикация мощности принятого сигнала физического уровня , если доступно, в противном случае None . Статистика физического уровня должна быть включена на канале, чтобы этот метод возвращал значение.

`get_snr` ()

Возвращает

Отношение сигнал/шум физического уровня , если доступно, в противном случае None . Статистика физического уровня должна быть включена на канале, чтобы этот метод возвращал значение.

`get_q` ()

Возвращает

Качество физического уровня *Link Quality* , если доступно, иначе `None` . Статистика физического уровня должна быть включена на канале, чтобы этот метод возвращал значение.

`get_establishment_rate` ()

Возвращает

Скорость передачи данных, при которой произошла процедура установления соединения, в битах в секунду.

`get_mtu` ()

Возвращает

MTU установленного соединения.

`get_mdu` ()

Возвращает

MDU пакета установленного соединения.

`get_expected_rate` ()

Возвращает

Ожидаемая скорость передачи данных пакета в пути для установленного соединения.

`get_mode` ()

Возвращает

Режим установленного соединения.

`get_age` ()

Возвращает

Время в секундах с момента установки данного соединения.

`no_inbound_for` ()

Возвращает

Время в секундах с момента получения последнего входящего пакета по соединению. Это включает пакеты keepalive.

no_outbound_for ()

Возвращает

Время в секундах с момента отправки последнего исходящего пакета по каналу. Это включает пакеты keepalive.

no_data_for ()

Возвращает

Время в секундах с момента передачи данных полезной нагрузки по каналу. Это исключает keepalive-пакеты.

inactive_for ()

Возвращает

Время в секундах с момента активности по каналу. Это включает пакеты keepalive.

get_remote_identity ()

Возвращает

Идентификатор удаленного узла, если он известен. Вызов этого метода не будет запрашивать у удаленного инициатора раскрытие его идентификатора. Возвращает `None`, если инициатор канала ещё не вызвал независимо метод `identify`(identity).

teardown()

Закрывает канал и очищает ключи шифрования. Новые ключи будут использоваться, если будет установлено новое соединение с тем же пунктом назначения.

get_channel ()

Получить `Channel`для этого канала.

Возвращает

объект`Channel`

set_link_closed_callback (*callback*)

Регистрирует функцию, которая будет вызвана при разрыве канала.

Параметры

callback – Функция или метод с сигнатурой*callback(link)*,который будет вызван.

set_packet_callback(*callback*)

Регистрирует функцию, которая будет вызвана при получении пакета по данной ссылке.

Параметры

callback – Функция или метод с сигнатурой*callback(message, packet)*для вызова.

set_resource_callback(*callback*)

Регистрирует функцию, которая будет вызвана при объявлении ресурса по данной ссылке. Если функция возвращает *True*, ресурс будет принят. Если она возвращает *False*, он будет проигнорирован.

Параметры

callback – Функция или метод с сигнатурой*callback(resource)*для вызова.
Обратите внимание, что в настоящее время доступна только основная информация о ресурсе, такая как *get_transfer_size()*, *get_data_size()*, *get_parts()* и *is_compressed()*.

set_resource_started_callback(*callback*)

Регистрирует функцию, которая будет вызвана при начале передачи ресурса по данной ссылке.

Параметры

callback – Функция или метод с сигнатурой*callback(resource)*для вызова.

set_resource_concluded_callback(*callback*)

Регистрирует функцию, которая будет вызвана по завершении передачи ресурса по данной ссылке.

Параметры

callback – Функция или метод с сигнатурой*callback(resource)*для вызова.

set_remote_identified_callback(*callback*)

Регистрирует функцию, которая будет вызвана, когда инициирующий пир идентифицирован по этой ссылке.

Параметры

callback – Функция или метод с сигнатурой*callback(link, identity)*,которые должны быть вызваны.

set_resource_strategy(*resource_strategy*)

Устанавливает стратегию ресурсов для ссылки.

Параметры

resource_strategy – Одна из RNS.Link.ACCEPT_NONE , RNS.Link.ACCEPT_ALL или RNS .Link.ACCEPT_APP . Если RNS.Link.ACCEPT_APP установлено, *resource_callback* будет вызвана для определения, должен ли ресурс быть принят или нет.

Вызывает

TypeError , если стратегия ресурсов не поддерживается.

10.7 Подтверждение запроса

class RNS.**RequestReceipt**

Экземпляр этого класса возвращается методом request экземпляров RNS.Link . Он никогда не должен быть создан вручную. Он предоставляет методы для проверки статуса, времени отклика и данных ответа по завершении запроса.

get_request_id()

Возвращает

Идентификатор запроса как *bytes*.

get_status ()

Возвращает

Текущий статус запроса, один из RNS.RequestReceipt.FAILED , RNS. RequestReceipt.SENT , RNS.RequestReceipt.DELIVERED , RNS.RequestReceipt. READY .

get_progress ()

Возвращает

Прогресс получения ответа в виде *float* в диапазоне от 0.0 до 1.0.

get_response()

Возвращает

Ответ в виде *bytes*, если он готов, в противном случае *None*.

get_response_time ()

Возвращает

Время ответа на запрос в секундах.

concluded()

Возвращает

True, если связанный запрос завершён (успешно или с ошибкой), в противном случае False.

10.8 Ресурс

class RNS.**Resource**(*data*, *link*, *advertise=True*, *auto_compress=True*, *callback=None*, *progress_callback=None*, *timeout=None*)

Класс Resource позволяет передавать произвольные объёмы данных по каналу связи. Он автоматически обрабатывает упорядочивание, сжатие, координацию и проверку контрольных сумм.

Параметры

- **data** – Данные для передачи. Может быть *bytes* или открытым *file handle* . Подробности см. в *Filetransfer Example* .
- **link** – Экземпляр *RNS.Link*, по которому будут передаваться данные.
- **advertise** – Необязательный параметр. Определяет, следует ли автоматически анонсировать ресурс. Может принимать значения *True* или *False* .
- **auto_compress** – Необязательный параметр. Определяет, следует ли автоматически сжимать ресурс. Может принимать значения *True* или *False*.
- **callback** – Необязательный *callable* с сигнатурой *callback(resource)* . Будет вызван после завершения передачи ресурса.
- **progress_callback** – Необязательный *callable* с сигнатурой *callback(resource)* . Будет вызываться каждый раз при обновлении прогресса передачи ресурса.

advertise()

Анонсирует ресурс. Если другая сторона соединения принимает анонс ресурса, начнется его передача.

cancel()

Отменяет передачу ресурса.

get_progress()

Возвращает

Текущий прогресс передачи ресурса в виде *float* от 0.0 до 1.0.

get_transfer_size()

Возвращает

Количество байтов, необходимых для передачи ресурса.

get_data_size()

Возвращает

Общий размер данных ресурса.

get_parts()

Возвращает

Количество частей, на которые будет разбит ресурс для передачи.

get_segments()

Возвращает

Количество сегментов, на которые разделен ресурс.

get_hash()

Возвращает
Хэш ресурса.

is_compressed()

Возвращает
Сжат ли ресурс.

10.9 Канал

class RNS.Channel.**Channel**

Обеспечивает надежную доставку сообщений по каналу связи.

Каналотличается от Requestи Resourceнесколькими важными способами:

Непрерывный
Сообщения могут быть отправлены или получены, пока Linkоткрыт.

Двунаправленный
Сообщения могут быть отправлены в любом направлении по Link; ни один из концов не является клиентом или сервером.

Ограниченный размером
Сообщения должны быть закодированы в один пакет.

Channel похож на Packet , за исключением того, что он обеспечивает надежную доставку (автоматические повторные попытки), а также структуру для обмена несколькими типами сообщений по Link .

Channel не инстанцируется напрямую, а получается из Linkс помощьюget_channel().

register_message_type (*message_class: Type[* MessageBase *]*) Регистрирует класс сообщений для приёма через Channel .

Классы сообщений должны расширятьMessageBase.

Параметры
message_class – Класс для регистрации

add_message_handler(*callback: MessageCallbackType*)

Добавляет обработчик для входящих сообщений. Обработчик имеет следующую сигнатуру:

```
(message: MessageBase) -> bool
```

Обработчики обрабатываются в порядке их добавления. Если какой-либо обработчик возвращает True, обработка сообщения прекращается; обработчики после возвращающего обработчика не будут вызваны.

Параметры
callback – Вызываемая функция

remove_message_handler(*callback: MessageCallbackType*)

Удаляет обработчик, добавленный с помощьюadd_message_handler.

Параметры
callback – Обработчик для удаления

is_ready_to_send() → bool

Проверяет, готов ли Channelк отправке.

Возвращает
True, если готов

send(*message:* MessageBase) → Envelope

Отправить сообщение. Если предпринята попытка отправки сообщения, а Channelне готов, генерируется исключение.

Параметры
message – экземпляр подкласса MessageBase

property mdu

Максимальная единица данных: количество байтов, доступных сообщению для использования за одну отправку. Это значение корректируется из Link MDU для учёта информации заголовка сообщения.

Возвращает
количество доступных байтов

10.10 MessageBase

class RNS.**MessageBase**

Базовый тип для любых сообщений, отправляемых или получаемых по Channel. Подклассы должны определять два абстрактных метода, а также переменную класса MSGTYPE .

MSGTYPE = None

Определяет уникальный идентификатор для класса сообщения.

- Должен быть уникальным для всех классов, зарегистрированных в Channel
- Должно быть меньше 0xf000. Значения, больше или равные 0xf000, зарезервированы.

abstractmethod pack() → bytes

Создаёт и возвращает бинарное представление сообщения

Returns
двоичное представление сообщения

abstractmethod unpack(*raw: bytes*)

Заполняет сообщение из двоичного представления

Параметры
raw – двоичное представление

10.11 Буфер

class RNS.**Buffer**

Статические функции для создания буферизованных потоков, которые отправляют и получают данные через Channel.

Эти функции используют BufferedReader, BufferedWriter и BufferedRWPair для добавления буферизации к RawChannelReader и RawChannelWriter.

static create_reader(*stream_id: int*, *channel:* Channel, *ready_callback: Callable[[int], None] | None = None*) → BufferedReader

Создает буферизованный reader, который читает бинарные данные, отправленные через Channel , с опциональным коллбэком при доступности новых данных.

Сигнатура коллбэка:(ready_bytes: int) -> None

Для получения дополнительной информации о функциях этого объекта, специфичных для чтения, см. документацию Python для BufferedReader

Параметры

- **stream_id** – локальный идентификатор потока для приема
- **channel** – канал для приема
- **ready_callback** – функция, вызываемая при появлении новых данных

Возвращает
объект BufferedReader

static create_writer(*stream_id: int*, *channel:* Channel) → BufferedWriter

Создает буферизованный записывающий модуль, который записывает бинарные данные через `Channel`.

Для получения дополнительной информации о функциях этого объекта, специфичных для записи, см. документацию Python по `BufferedWriter`

Параметры

- **stream_id** – удаленный идентификатор потока для отправки
- **channel** – канал для отправки

Возвращает
объект BufferedWriter

static create_bidirectional_buffer(*receive_stream_id: int*, *send_stream_id: int*, *channel:* Channel, *ready_callback: Callable[[int], None] | None = None*) → BufferedRWPair

Создает буферизованную пару для чтения/записи, которая считывает и записывает бинарные данные через `Channel`, с необязательным обратным вызовом при появлении новых данных.

Сигнатура коллбэка:`(ready_bytes: int) -> None`

Для получения дополнительной информации о функциях этого объекта, специфичных для чтения, см. документацию Python по

`BufferedRWPair` Параметры

- **receive_stream_id** – локальный идентификатор потока для приёма
- **send_stream_id** – удаленный идентификатор потока для отправки
- **channel** – канал для отправки и приёма
- **ready_callback** – функция, вызываемая при появлении новых данных

Возвращает
объект BufferedRWPair

10.12 RawChannelReader

class RNS.**RawChannelReader**(*stream_id: int*, *channel:* Channel)

Реализация RawIOBase, которая принимает данные бинарного потока, отправленные через `Channel`.

Этот класс, как правило, не требует прямого инстанцирования. Используйте функции *RNS.Buffer.create_reader()*, *RNS.Buffer.create_writer()* и *RNS.Buffer.create_bidirectional_buffer()* для создания буферизованных потоков с опциональными коллбэками.

Для получения дополнительной информации об API этого объекта см. документацию Python по `RawIOBase`.

__init__(*stream_id: int*, *channel:* Channel)

Создать Raw-читатель канала.

Параметры

- **stream_id** – локальный идентификатор потока для приёма
- **channel** – объект Channelдля приёма данных

add_ready_callback(*cb: Callable[[int], None]*)

Добавить функцию, которая будет вызываться при доступности новых данных. Функция должна иметь сигнатуру `(ready_bytes: int) -> None`

Параметры
cb – функция для вызова

remove_ready_callback(*cb: Callable[[int], None]*)

Удалить функцию, добавленную с помощью*RNS.RawChannelReader.add_ready_callback()*

Параметры
cb – функция для удаления

10.13 RawChannelWriter

class RNS.**RawChannelWriter**(*stream_id: int*, *channel:* Channel)

Реализация RawIOBase, которая принимает данные бинарного потока, отправленные по каналу.

Этот класс, как правило, не требует прямого инстанцирования. Используйте функции *RNS.Buffer.create_reader()* , *RNS.Buffer.create_writer()* и *RNS.Buffer.create_bidirectional_buffer()* для создания буферизованных потоков с опциональными коллбэками.

Для получения дополнительной информации об API этого объекта см. документацию Python по `RawIOBase`.

__init__(*stream_id: int*, *channel:* Channel)

Создать объект для записи в необработанный канал.

Параметры

- **stream_id** – идентификатор удаленного потока для отправки
- **channel** – объект Channelдля отправки

10.14 Транспорт

class RNS.**Transport**

С помощью статических методов этого класса можно взаимодействовать с транспортной системой Reticulum.

PATHFINDER_M = 128

Максимальное количество переходов, через которое Reticulum будет транспортировать пакет.

static register_announce_handler(*handler*)

Регистрирует обработчик объявлений.

Параметры
handler – *Должен быть объектом с атрибутом aspect_filter и вызываемым методом* re- ceived_announce(destination_hash, announced_identity, app_data) *или* received_announce(destination_hash, announced_identity, app_data, announce_packet_hash) *или* received_announce(destination_hash, announced_identity, app_data, an- nounce_packet_hash, is_path_response). *Может опционально иметь атрибут* receive_path_responses , *установленный в* True , *чтобы также получать все ответы на пути, в дополнение к активным объявлениям. См.* Announce Example *для получения дополнительной информации.*

static deregister_announce_handler (*handler*) Отменяет регистрацию обработчика объявлений.

Параметры
handler – Обработчик объявлений, который необходимо отменить.

static has_path (*destination_hash*)

Параметры
destination_hash – Хеш назначения в формате *bytes*.

Возвращает
True , если путь к месту назначения известен, в противном случае *False*.

static hops_to (*destination_hash*)

Параметры
destination_hash – Хеш назначения в формате *bytes*.

Возвращает
Количество переходов до указанного пункта назначения, или `RNS.Transport.PATHFINDER_M`, если количество переходов неизвестно.

static next_hop (*destination_hash*)

Параметры
destination_hash – Хеш назначения в формате *bytes*.

Возвращает
Хэш назначения в виде *bytes* для следующего перехода к указанному пункту назначения, или *None*, если следующий переход неизвестен.

static next_hop_interface (*destination_hash*)

Параметры
destination_hash – Хеш назначения в формате *bytes*.

Возвращает
Интерфейс для следующего перехода к указанному пункту назначения, или *None*,если интерфейс неизвестен.

static request_path (*destination_hash* , *on_interface=None* , *tag=None* , *recursive=False*) Запрашивает путь к пункту назначения из сети. Если другой достижимый пир в сети знает путь, он объявит его.

Параметры

- **destination_hash** – Хеш назначения в формате *bytes*.
- **on_interface** – Если указано, запрос пути будет отправлен только через этот интерфейс. При обычном использовании Reticulum обрабатывает это автоматически, и этот параметр не должен использоваться.

ИНДЕКС

Символы

__init__() (*метод RNS.RawChannelReader*), 175
__init__() (*метод RNS.RawChannelWriter*), 176

A

accepts_links() (*метод RNS.Destination*), 163
add_message_handler() (*метод RNS.Channel.Channel*), 173
add_ready_callback() (*метод RNS.RawChannelReader*), 176
advertise() (*метод RNS.Resource*), 172
announce() (*метод RNS.Destination*), 162
ANNOUNCE_CAP (*атрибут RNS.Reticulum*), 157
app_and_aspects_from_name() (*статический метод RNS.Destination*), 162

B

Buffer (*класс в RNS*), 174

C

cancel() (*метод RNS.Resource*), 172
Channel (*класс в RNS.Channel*), 173
clear_default_app_data() (*RNS.Destination method*), 166
concluded() (*RNS.RequestReceipt method*), 171
create_bidirectional_buffer() (*RNS.Buffer static method*), 175
create_keys() (*RNS.Destination method*), 165
create_reader() (*RNS.Buffer static method*), 174
create_writer() (*RNS.Buffer static method*), 175
current_ratchet_id() (*RNS.Identity static method*), 160
CURVE (*RNS.Identity attribute*), 159
CURVE (*RNS.Link attribute*), 168

D

decrypt() (*RNS.Destination method*), 165
decrypt() (*RNS.Identity method*), 161
deregister_announce_handler() (*RNS.Transport static method*), 176
deregister_request_handler() (*RNS.Destination method*), 164
Destination (*класс в RNS*), 162

E

enable_ratchets() (*метод RNS.Destination*), 164
encrypt() (*метод RNS.Destination*), 165
encrypt() (*метод RNS.Identity*), 161
ENCRYPTED_MDU (*атрибут RNS.Packet*), 166
enforce_ratchets() (*метод RNS.Destination*), 164
ESTABLISHMENT_TIMEOUT_PER_HOP (*атрибут RNS.Link*), 168
expand_name() (*статический метод RNS.Destination*), 162

F

from_bytes() (*статический метод RNS.Identity*), 160
from_file() (*статический метод RNS.Identity*), 160
full_hash() (*статический метод RNS.Identity*), 159

G

get_age() (*метод RNS.Link*), 169
get_channel() (*метод RNS.Link*), 170
get_data_size() (*метод RNS.Resource*), 172
get_establishment_rate() (*RNS.Link метод*), 169
get_expected_rate() (*RNS.Link метод*), 169
get_hash() (*RNS.Resource метод*), 172
get_instance() (*RNS.Reticulum статический метод*), 158
get_mdu() (*RNS.Link метод*), 169
get_mode() (*RNS.Link метод*), 169
get_mtu() (*RNS.Link метод*), 169
get_parts() (*RNS.Resource метод*), 172
get_private_key() (*RNS.Destination метод*), 165
get_private_key() (*RNS.Identity метод*), 160
get_progress() (*RNS.RequestReceipt метод*), 171
get_progress() (*RNS.Resource метод*), 172
get_public_key() (*RNS.Identity метод*), 160
get_q() (*RNS.Link метод*), 169
get_q() (*RNS.Packet метод*), 167
get_random_hash() (*RNS.Identity static method*), 160
get_remote_identity() (*RNS.Link method*), 170
get_request_id() (*RNS.RequestReceipt method*), 171
get_response() (*RNS.RequestReceipt method*), 171
get_response_time() (*RNS.RequestReceipt method*), 171

get_rssi() (*RNS.Link method*), 169
get_rssi() (*RNS.Packet method*), 166
get_rtt() (*RNS.PacketReceipt method*), 167
get_segments() (*RNS.Resource method*), 172
get_snr() (*RNS.Link method*), 169
get_snr() (*RNS.Packet method*), 166
get_status() (*RNS.PacketReceipt method*), 167
get_status() (*RNS.RequestReceipt method*), 171
get_transfer_size() (*RNS.Resource method*), 172

H

has_path() (*RNS.Transport static method*), 177
hash() (*статический метод RNS.Destination*), 162
hash_from_name_and_identity() (*статический метод RNS.Destination*), 162
hops_to() (*статический метод RNS.Transport*), 177

I

identify() (*метод RNS.Link*), 168
Identity (*класс в RNS*), 158
inactive_for() (*метод RNS.Link*), 170
is_compressed() (*метод RNS.Resource*), 173
is_ready_to_send() (*метод RNS.Channel.Channel*), 173

K

KEEPALIVE (*атрибут RNS.Link*), 168
KEEPALIVE_TIMEOUT_FACTOR (*атрибут RNS.Link*), 168
KEYSIZE (*атрибут RNS.Identity*), 159

L

Link (*класс в RNS*), 167
LINK_MTU_DISCOVERY (*атрибут RNS.Reticulum*), 157
link_mtu_discovery() (*статический метод RNS.Reticulum*), 158
load_private_key() (*метод RNS.Destination*), 165
load_private_key() (*метод RNS.Identity*), 161
load_public_key() (*метод RNS.Identity*), 161

M

mdu (*свойство RNS.Channel.Channel*), 174
MessageBase (*класс в RNS*), 174
MINIMUM_BITRATE (*атрибут RNS.Reticulum*), 158
MSGTYPE (*атрибут RNS.MessageBase*), 174
MTU (*атрибут RNS.Reticulum*), 157

H

next_hop() (*статический метод RNS.Transport*), 177
next_hop_interface() (*статический метод RNS.Transport*), 177
no_data_for() (*метод RNS.Link*), 170
no_inbound_for() (*метод RNS.Link*), 169
no_outbound_for() (*метод RNS.Link*), 169

П

pack() (*метод RNS.MessageBase*), 174
Packet (*класс в RNS*), 166
PacketReceipt (*класс в RNS*), 167
PATHFINDER_M (*атрибут RNS.Transport*), 176
PLAIN_MDU (*атрибут RNS.Packet*), 166

R

RATCHET_COUNT (*атрибут RNS.Destination*), 162
RATCHET_EXPIRY (*атрибут RNS.Identity*), 159
RATCHET_INTERVAL (*атрибут RNS.Destination*), 162
RATCHETSIZE (*атрибут RNS.Identity*), 159
RawChannelReader (*класс в RNS*), 175
RawChannelWriter (*класс в RNS*), 176
recall() (*статический метод RNS.Identity*), 159
recall_app_data() (*статический метод RNS.Identity*), 159
register_announce_handler() (*статический метод RNS.Transport*), 176
register_message_type() (*метод RNS.Channel.Channel*), 173
register_request_handler() (*RNS.Destination method*), 163
remote_management_enabled() (*RNS.Reticulum static method*), 158
remove_message_handler() (*RNS.Channel.Channel method*), 173
remove_ready_callback() (*RNS.RawChannelReader method*), 176
request() (*RNS.Link method*), 168
request_path() (*RNS.Transport static method*), 177
RequestReceipt (*class in RNS*), 171
resend() (*RNS.Packet method*), 166
Resource (*class in RNS*), 172
Reticulum (*class in RNS*), 157

S

send() (*RNS.Channel.Channel method*), 173
send() (*RNS.Packet method*), 166
set_default_app_data() (*RNS.Destination method*), 165
set_delivery_callback() (*RNS.PacketReceipt method*), 167
set_link_closed_callback() (*RNS.Link method*), 170
set_link_established_callback() (*RNS.Destination method*), 163
set_packet_callback() (*RNS.Destination method*), 163
set_packet_callback() (*RNS.Link method*), 170
set_proof_requested_callback() (*RNS.Destination method*), 163
set_proof_strategy() (*RNS.Destination method*), 163

set_ratchet_interval() (*RNS.Destination method*), 164
set_remote_identified_callback() (*RNS.Link method*), 171
set_resource_callback() (*RNS.Link method*), 170
set_resource_concluded_callback() (*RNS.Link method*), 170
set_resource_started_callback() (*RNS.Link method*), 170
set_resource_strategy() (*RNS.Link method*), 171
set_retained_ratchets() (*RNS.Destination method*), 164
set_timeout() (*RNS.PacketReceipt method*), 167
set_timeout_callback() (*RNS.PacketReceipt method*), 167
should_use_implicit_proof() (*RNS.Reticulum static method*), 158
sign() (*RNS.Destination method*), 165
sign() (*RNS.Identity method*), 161
STALE_GRACE (*RNS.Link attribute*), 168
STALE_TIME (*RNS.Link attribute*), 168

T

teardown() (*RNS.Link method*), 170
to_file() (*RNS.Identity method*), 160
track_phy_stats() (*RNS.Link method*), 168
Transport (*класс в RNS*), 176
transport_enabled() (*RNS.Reticulum static method*), 158
truncated_hash() (*RNS.Identity static method*), 159
TRUNCATED_HASHLENGTH (*RNS.Identity attribute*), 159

U

unpack() (*RNS.MessageBase method*), 174

V

validate() (*RNS.Identity method*), 161