

Содержание

Предисловие	12
Глава 1. Тенденции в современной промышленной эксплуатации сетей	20
Возникновение технологии программно определяемой сети	20
OpenFlow	21
Что такое программно определяемая сеть	25
Резюме	39
Глава 2. Автоматизация сети	40
Для чего нужна автоматизация сети	40
Упрощение архитектуры	41
Детерминированные результаты	42
Гибкость бизнеса	42
Типы автоматизации сети	43
Подготовка и настройка устройств	43
Сбор данных	46
Переходы между платформами	47
Управление конфигурацией	49
Совместимость	49
Составление отчетов	50
Устранение проблем	51
Развитие уровня управления от протокола SNMP до API устройств	53
Прикладные программные интерфейсы (API)	53
Влияние концепции открытых сетей	57
Автоматизация сети в эпоху SDN	58
Резюме	59
Глава 3. Операционная система Linux	60
Изучение ОС Linux с точки зрения автоматизации сети	60
Краткая история создания ОС Linux	61
Дистрибутивы Linux	62
Red Hat Enterprise Linux, Fedora и CentOS	62
Debian, Ubuntu и другие производные дистрибутивы	64
Другие дистрибутивы Linux	66
Работа в ОС Linux	66
Перемещение по файловой системе	67
Работа с файлами и каталогами	72
Выполнение программ	79

Работа с демонами	82
Работа с сетями в ОС Linux	87
Работа с интерфейсами	87
Маршрутизация для конечного хоста.....	98
Конфигурация маршрутизатора	103
Коммутация.....	105
Резюме.....	111

Глава 4. Изучение языка программирования Python

для применения в сетевой среде	112
Должны ли сетевые инженеры уметь писать программный код?	113
Использование интерактивного интерпретатора Python	115
Типы данных языка Python.....	117
Использование строк	118
Использование числовых значений	128
Использование логических значений	130
Использование списков	133
Использование словарей	138
Множества и кортежи языка Python	143
Использование условных логических выражений.....	145
Концепция объекта, содержащего другие объекты.....	148
Использование циклов.....	149
Использование цикла while.....	149
Использование цикла for	150
Использование функций.....	154
Работа с файлами.....	158
Чтение данных из файла	158
Запись данных в файл.....	161
Создание программ на языке Python.....	163
Создание простого скрипта на языке Python.....	163
Что такое shebang	164
Перемещение кода из интерпретатора Python в независимый скрипт	166
Работа с модулями языка Python	167
Передача аргументов в скрипт	169
Использование pip для установки пакетов языка Python.....	171
Советы, приемы и дополнительная информация по использованию языка Python	173
Резюме.....	179

Глава 5. Форматы и модели данных

Введение в форматы данных	180
Типы данных	182
YAML	184
Краткий обзор основ YAML	184
Работа с YAML в коде Python	187
Модели данных в YAML	188
XML	190

Основы XML.....	190
Использование определения схемы XML Schema Definition (XSD) для моделей данных	191
Преобразование XML с помощью XSLT	193
Поиск в данных XML с использованием XQuery.....	197
JSON	197
Основы формата JSON	198
Обработка формата JSON в коде Python.....	200
Использование механизма JSON Schema для моделей данных.....	201
Создание моделей данных с использованием YANG	202
Общий обзор языка YANG.....	202
Практическое применение языка YANG	203
Резюме.....	207
Глава 6. Шаблоны сетевой конфигурации	208
Современные языки шаблонов.....	209
Использование шаблонов для веб-разработки.....	210
Универсальность шаблонов.....	211
Важность использования шаблонов в процессе автоматизации сети	212
Язык Jinja для создания шаблонов сетевой конфигурации	213
Почему именно Jinja	213
Динамическая вставка данных в простой шаблон Jinja	214
Обработка файла шаблона Jinja средствами языка Python.....	215
Условные выражения и циклы	217
Фильтры Jinja.....	224
Наследование шаблонов в языке Jinja	227
Создание переменных в Jinja	228
Резюме.....	229
Глава 7. Использование сетевых прикладных программных интерфейсов (API).....	230
Основы сетевых API.....	231
Введение в API-интерфейсы на основе протокола HTTP.....	231
Основы NETCONF	236
Практическое использование сетевых API	244
Практическое использование API на основе протокола HTTP	245
Практическое использование NETCONF	252
Автоматизация с использованием сетевых API	261
Использование библиотеки requests	262
Использование Python-библиотеки ncclient	292
Использование библиотеки netmiko.....	317
Резюме.....	322
Глава 8. Управление исходным кодом с помощью Git.....	325
Варианты использования средств управления исходным кодом	326
Преимущества системы управления исходным кодом	326

Отслеживание изменений.....	327
Учетные записи.....	327
Процесс и рабочий поток.....	327
Преимущества системы управления исходным кодом в сетевой среде.....	328
Знакомство с Git.....	328
Краткая история создания и развития Git.....	329
Терминология Git.....	330
Обзор архитектуры Git.....	331
Работа с системой Git.....	332
Установка системы Git.....	332
Создание репозитория.....	333
Добавление файлов в репозиторий.....	333
Выполнение коммита изменений в репозиторий.....	335
Внесение изменений и выполнение коммитов в отслеживаемые файлы.....	339
Отмена фиксации файлов в индексе.....	342
Исключение файлов из репозитория.....	345
Получение более подробной информации о репозитории.....	349
Определение различий между версиями файлов.....	354
Создание ветвей версий в системе Git.....	358
Создание ветви.....	363
Выбор активной ветви.....	364
Объединение и удаление ветвей.....	366
Совместная работа группы сотрудников в системе Git.....	371
Совместная работа в нескольких системах, использующих Git.....	372
Совместная работа с использованием онлайн-сервисов на основе Git.....	389
Резюме.....	395
Глава 9. Инструментальные средства автоматизации.....	396
Краткий обзор инструментальных средств автоматизации.....	396
Использование Ansible.....	399
Основы работы Ansible.....	400
Создание inventory-файла.....	401
Выполнение сценария Ansible.....	408
Использование файлов переменных.....	412
Создание комплектов сценариев Ansible для автоматизации сети.....	414
Использование сторонних модулей Ansible от независимых авторов.....	433
Резюме по системе Ansible.....	436
Автоматизация сети с использованием Salt.....	437
Основы архитектуры Salt.....	437
Общая информация о Salt.....	440
Управление сетевыми конфигурациями с помощью Salt.....	458
Удаленное выполнение функций Salt.....	467
Управляемая событиями инфраструктура Salt.....	469
Дополнительная информация о Salt.....	475
Краткий итоговый обзор системы Salt.....	478
Автоматизация сети, управляемая событиями, с использованием	
StackStorm.....	479
Основные концепции системы StackStorm.....	480

Архитектура StackStorm	482
Операции и рабочие потоки	484
Сенсоры и триггеры	493
Правила	496
Краткий итоговый обзор системы StackStorm	499
Резюме	499
Глава 10. Непрерывная интеграция	500
Важные предпосылки	502
Чем проще, тем лучше	502
Люди, процесс и технология	503
Изучение программногo кода	503
Введение в непрерывную интеграцию	504
Основы непрерывной интеграции	504
Непрерывная доставка	506
Разработка через тестирование	508
Применимость методики непрерывной интеграции к сетевой среде	511
Конвейер непрерывной интеграции для сетевой среды	512
Рецензирование коллегами	513
Автоматизация сборки	519
Среда тестирования/разработки/перемещения данных	524
Инструментальные средства развертывания	528
Инструментальные средства тестирования и автоматизация сети по методике разработки через тестирование	531
Резюме	533
Глава 11. Формирование культуры автоматизации сети	535
Организационная стратегия и гибкость	536
Преобразование организации старого образца	536
Важность поддержки со стороны руководства	538
Купить или создать самостоятельно	540
Восприятие ситуаций критических сбоев	541
Практические навыки и обучение	543
Изучайте неизвестное	544
Сосредоточьтесь на основных принципах	545
Нужны ли сертификации?	546
Может ли автоматизация лишить людей работы	547
Резюме	548
Приложение А. Профессиональное управление сетевой средой в ОС Linux	550
Использование интерфейсов macvlan	550
Варианты практического использования интерфейсов macvlan	551
Создание, конфигурирование и удаление интерфейсов macvlan	551
Виртуальные машины в сетевой среде	553
Использование шлюза	554

Использование интерфейсов macvtap.....	557
Работа с сетевыми пространствами имен.....	558
Практические примеры использования сетевых пространств имен.....	559
Создание и удаление сетевых пространств имен.....	560
Размещение интерфейсов в сетевом пространстве имен.....	560
Выполнение команд в определенном сетевом пространстве имен.....	562
Соединение сетевых пространств имен с помощью пар veth.....	564
Использование контейнеров Linux в сетевой среде.....	566
Конфигурирование сетевой среды в LXC.....	567
Конфигурирование сетевой среды в Docker.....	568
Использование Open vSwitch.....	570
Установка OVS.....	570
Конфигурирование OVS.....	572
Соединение нескольких типов рабочих нагрузок в OVS.....	575
Приложение Б. Использование NAPALM.....	583
Управление конфигурацией с использованием NAPALM.....	583
Выполнение операции замены конфигурации.....	584
Выполнение операции объединения конфигураций.....	588
Получение данных от устройств с помощью NAPALM.....	591
Возможности интеграции NAPALM с другим ПО.....	593
Использование NAPALM в Ansible.....	594
Использование NAPALM в Salt.....	595
Использование NAPALM в StackStorm.....	596
Предметный указатель.....	598

Предисловие

Приветствуем всех читателей книги «Программно управляемые сети и их автоматизация».

Изменения в области промышленной эксплуатации сетей постоянны и значительны. Ориентация организаций и специалистов-профессионалов по сетям на восприятие идей и концепций сетевой программируемости и автоматизации сейчас приобретает гораздо большее значение, чем раньше, и стимулируется непрерывным процессом появления новых протоколов, новых технологий, новых моделей доставки, а также потребностями бизнес-процессов в плане большей интеллектуальности и гибкости для обеспечения конкурентоспособности. Но что означает сетевая программируемость и автоматизация? Начнем эту книгу с попытки найти возможности ответа на данный вопрос.

О ЧЕМ ЭТА КНИГА

В соответствии с названием главной темой книги является сетевая программируемость (программно управляемые сети) и автоматизация. По существу, сетевая программируемость и автоматизация практически является синонимом упрощения задач по конфигурированию, управлению и эксплуатации сетевого оборудования, сетевых топологий и поддержки сетевых соединений. При этом используется множество разнообразных компонентов, в том числе операционные системы, которые в настоящее время гораздо шире применяются в сетевых средах, чем в прошлом, новые методики, например методика непрерывной интеграции (*continuous integration*), и привлечение инструментальных средств, прежде относящихся только к арсеналу системного администратора (например, средства управления версиями исходного кода и системы управления конфигурацией). Мы полагаем, что все перечисленные компоненты играют важную роль в базовом определении сущности сетевой программируемости и автоматизации, поэтому рассматриваем все эти темы. Цель данной книги – позволить читателям получить основы знаний о сетевой программируемости и автоматизации.

КАК ОРГАНИЗОВАНА ЭТА КНИГА

Эту книгу не обязательно штудировать последовательно, страницу за страницей, мы разделили ее на главы таким образом, чтобы читатель с легкостью находил наиболее интересные ему темы. Возможно, правильное всего будет начать с последовательного чтения первых трех глав, так как в них представлена общая информация, то есть фактически закладывается основа понимания

всей остальной части книги. Далее вы можете свободно переходить к темам, которые вам наиболее интересны или немедленно требуются для практической работы. Мы пытались сохранить относительную независимость глав друг от друга, но, как и в процессе применения любой технологии, это оказалось не всегда выполнимо. Везде, где это возможно, мы приводим перекрестные ссылки, чтобы помочь читателям быстро находить необходимую информацию.

Ниже приводится краткий обзор организации книги по главам:

- глава 1 «Тенденции в современной промышленной эксплуатации сетей» представляет обзор основных событий и тенденций, способствовавших появлению программно определяемых сетей (software defined networks, SDN). Как вы увидите в главе 1, программно определяемая (или конфигурируемая) сеть стала главной первопричиной постоянно возрастающего сосредоточения внимания на сетевой программируемости и автоматизации;
- глава 2 «Автоматизация сети» продолжает обсуждение программно определяемых сетей, начатое в главе 1, но здесь главное внимание уделяется автоматизации сети: истории развития, влиянию сетевой автоматизации на операционные (рабочие) модели (и влиянию операционных моделей на автоматизацию);
- глава 3 «ОС Linux» представляет общий обзор операционной системы Linux. Эта глава не является полным описанием ОС Linux, ее основная задача – быстро ознакомить специалистов по сетям с рабочей средой Linux, с основным набором команд и с сетевыми концепциями, реализованными в Linux;
- глава 4 «Изучение языка программирования Python в сетевом контексте» знакомит специалистов по сетям с языком программирования Python (<http://python.org/>). Язык Python часто применяется для программируемых сетей и их автоматизации, поэтому в главе рассматриваются многие основные аспекты программирования на Python: типы данных, условные выражения, циклы, работа с файлами, функции, классы и модули;
- глава 5 «Форматы и модели данных» представляет наиболее распространенные форматы данных, которые часто встречаются в проектах автоматизации сетей. Рассматриваются языки обработки данных JavaScript Object Notation (JSON), eXtensible Markup Language (XML), YAML Ain't Markup Language (YAML). Затем описываются концепции моделирования данных и приводится краткое введение в язык YANG, широко распространенный язык моделирования данных для сетевой среды;



Что означает термин «формат данных»

Если для вас это незнакомый термин, не смущайтесь. Формат данных (data format) – это всего лишь способ кодирования или инкапсуляции данных при передаче между двумя пунктами (например, при возврате данных в ответ на вызов функции прикладного программного интерфейса (API)). Форматы данных подробно рассматриваются в главе 5.

- глава 6 «Шаблоны конфигурации сети» – рассматривается использование языков описания шаблонов для создания конфигураций сетевых устройств. Основное внимание сосредоточено на языке шаблонов Jinja, так как он весьма удачно взаимодействует с языком программирования (ЯП) Python. Кроме того, рассматриваются еще два языка описания шаблонов: Maiko и ERB. Maiko интегрируется с ЯП Python, а ERB используется в основном совместно с ЯП Ruby;
- глава 7 «Работа с сетевыми прикладными программными интерфейсами (API)» – рассматривается роль прикладных программных интерфейсов (application programming interfaces, API) в обеспечении сетевой программируемости и автоматизации. Описываются основные термины и технологии, применяемые в API, а также применение некоторых наиболее распространенных API от конкретных производителей – API для устройств и API для контроллеров – с примерами, наглядно демонстрирующими возможности их использования для обеспечения сетевой программируемости и автоматизации;
- глава 8 «Управление исходными кодами с помощью Git» представляет систему Git (<https://git-scm.com>) – общеизвестное и широко используемое инструментальное средство управления исходными кодами. Подчеркивается важность управления исходными кодами, рассматриваются возможности использования такой системы с точки зрения сетевой программируемости и автоматизации. Также рассматривается работа с известными онлайн-сервисами, например GitHub (<https://github.com>);
- глава 9 «Инструментальные средства автоматизации» подробно описывает использование инструментальных средств автоматизации с открытым исходным кодом, таких как Ansible (<http://www.ansible.com/home>), Salt (<http://saltstack.com>) и StackStorm (<https://stackstorm.com>), и способы применения этих инструментов специально для обеспечения сетевой программируемости и автоматизации;
- глава 10 «Непрерывная интеграция» описывает концепции непрерывной интеграции (continuous integration, CI) и соответствующие основные инструментальные средства и технологии. Рассматривается применение методики разработки через тестирование (test-driven development, TDD), инструментов исследования и контроля, а также комплексных сред разработки, таких как Jenkins и Gerrit, приводится пример формирования потока сетевой автоматизации, включающий все элементы методики непрерывной интеграции;
- глава 11 «Формирование культуры автоматизации сети» объясняет, почему правильно сформированная культура чрезвычайно важна и почему культура является основополагающим элементом для автоматизации сети. Рассматривается процесс формирования такой культуры;
- приложение А «Дополнительная информация о поддержке сетевой среды в ОС Linux» продолжает обсуждение, начатое в главе 3, но с более подробными описаниями работы с сетевыми интерфейсами macvlan, орга-

низации сетей с помощью виртуальных машин (VM), работы с сетевыми пространствами имен и с сетями, использующими контейнеры Linux (включая и контейнеры Docker (<https://www.docker.com>)). Также рассматривается использование Open vSwitch (OVS) (<http://openvswitch.org>);

- приложение Б «Использование NAPALM» представляет введение в использование библиотеки NAPALM (Network Automation and Programmability Abstraction Layer with Multi-vendor support), написанной на ЯП Python. Рассматривается применение NAPALM для управления конфигурацией, не зависящей от конкретных производителей, а также для извлечения данных из сетевых устройств. Кратко описываются возможности интеграции NAPALM с инструментальными средствами Ansible, Salt и StackStorm, которые подробно рассматривались в главе 9.

Для кого предназначена эта книга

Как уже было сказано ранее, главная цель этой книги – предоставить читателям основополагающие знания и набор основных практических навыков в области сетевой программируемости и автоматизации. Мы надеемся, что представители различных профессий в области информационных технологий извлекут пользу из чтения нашей книги.

Сетевые инженеры

Вполне естественно, что большую часть читателей книги, посвященной сетевой программируемости и автоматизации, составляют «обычные» сетевые инженеры, то есть специалисты, достаточно хорошо разбирающиеся в сетевых протоколах, в конфигурации сетевых устройств и в вопросах функционирования и управления сетями. Мы надеемся, что книга поможет сетевым инженерам сделать свою повседневную работу более эффективной и производительной с помощью автоматизации и практического использования программируемости сетей.

Предварительные требования

Сетевым инженерам, заинтересованным в более глубоком изучении сетевой программируемости и автоматизации, не требуются какие-либо предварительные знания в области разработки ПО, программирования, автоматизации или инструментальных средств, связанных с DevOps. Единственное предварительное требование – свободное мышление и восприятие без предвзятости и желание изучать новые технологии и их влияние на деятельность специалистов по сетям и на всю сетевую индустрию в целом.

Системные администраторы

Системные администраторы, ответственные в основном за управление системами, подключенными к сетям, возможно, уже обладают некоторым предварительным уровнем знаний и практическим опытом использования инструмен-

тальных средств, рассматриваемых в книге (в особенности ОС Linux, систем управления исходным кодом и систем управления конфигурацией). Таким образом, книга может помочь им расширить знания и понимание функциональности таких инструментов, представляя их в различных контекстах (например, использование Ansible для конфигурирования сетевого коммутатора в сравнении с использованием Ansible для конфигурирования сервера, работающего под управлением дистрибутива Linux).

Предварительные требования

В этой книге не рассматриваются и не объясняются базовые сетевые протоколы и концепции. Но мы предполагаем, что многие системные администраторы, постоянно занимающиеся управлением подключенных к сетям систем, обладают достаточным уровнем знаний об основных сетевых протоколах, поэтому хорошо подготовлены к чтению. Если вы не вполне уверены в качестве своих знаний о работе в сетях, рекомендуем предварительно изучить одну из книг, подробно описывающих основные сетевые концепции и принципы функционирования. Например, неплохим вариантом является книга «Packet Guide to Core Network Protocols» издательства O'Reilly.

Разработчики программного обеспечения

Разработчики программного обеспечения также могут получить много полезной информации при чтении этой книги. Многие разработчики уже обладают опытом работы с языками программирования и средствами разработки, обсуждаемыми в книге (например, Python и Git). Как и для системных администраторов, для разработчиков может оказаться полезным рассмотрение использования средств разработки и языков программирования в применении к сетевым средам (например, описание возможностей Python для извлечения и сохранения специализированных сетевых данных).

Предварительные требования

Предполагается, что читатели обладают базовыми знаниями об основных сетевых протоколах и концепциях, поскольку все предлагаемые в книге примеры выполняются исключительно в сетевой среде. Если для разработчиков ПО сети являются новой темой, то, как и системным администраторам, мы предлагаем предварительно изучить одну из книг по основным сетевым концепциям.

ИНСТРУМЕНТЫ, ИСПОЛЬЗУЕМЫЕ В КНИГЕ

В области сетевой программируемости и автоматизации, как и в любой технической отрасли, существует множество версий и вариаций технологий и инструментальных средств. Поэтому мы придерживаемся общепризнанных стандартов при рассмотрении инструментальных средств в данной книге и выбираем самые лучшие, по нашему мнению, из этих инструментов, чтобы читатели сами оценили их полезность для своей практической деятельности. Например, при

наличии множества разнообразных дистрибутивов Linux основное внимание мы все же уделяем дистрибутивам Debian, Ubuntu (производному от Debian) и CentOS (производному от Red Hat Enterprise Linux (RHEL)). Чтобы облегчить восприятие информации для читателей, мы рассматриваем конкретную версию инструментального средства в каждой соответствующей главе.

ОНЛАЙН-РЕСУРСЫ

Мы понимаем, что в одной книге невозможно представить весь материал, полностью освещающий такую обширную тему, как сетевая программируемость и автоматизация. Поэтому на протяжении всей книги мы многократно ссылаемся на дополнительные онлайн-ресурсы, которые могут оказаться весьма полезными при более глубоком изучении концепций, принципов и практических навыков, рассматриваемых в этой книге.

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ И СОГЛАШЕНИЯ, ПРИНЯТЫЕ В КНИГЕ




В книге используются следующие типографские соглашения:

Курсив – используется для смыслового выделения важных положений, новых терминов, URL-адресов и адресов электронной почты в интернете, имен команд и утилит, а также имен и расширений файлов и каталогов.

Моноширинный шрифт – используется для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов, ключевых слов и других программных конструкций и элементов исходного кода.

Моноширинный полужирный шрифт – используется для обозначения команд или фрагментов текста, которые пользователь должен ввести дословно без изменений.

Моноширинный курсив – используется для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены соответствующими контексту реальными значениями.

-  Такая пиктограмма обозначает указание или примечание общего характера.
-  Эта пиктограмма обозначает предупреждение или особое внимание к потенциально опасным объектам.
-  Такая пиктограмма обозначает совет или рекомендацию.

ОТЗЫВЫ И ПОЖЕЛАНИЯ

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

СКАЧИВАНИЕ ИСХОДНОГО КОДА ПРИМЕРОВ

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

СПИСОК ОПЕЧАТОК

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг — возможно, ошибку в тексте или в коде, — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от огорчения и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

НАРУШЕНИЕ АВТОРСКИХ ПРАВ

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и O'Reilly очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

БЛАГОДАРНОСТИ

Создание этой книги было бы невозможным без помощи и поддержки большой группы людей.

Во-первых, мы хотели бы выразить свою благодарность чрезвычайно активному сообществу людей, объединенных идеей сетевой автоматизации. Назвать всех по именам невозможно, потому что их слишком много, но эти люди создали проекты с открытым исходным кодом, такие как NAPALM и Netmiko, всячески способствовали вовлечению новичков в процесс обучения сетевой автоматизации и постоянно делились своими знаниями и опытом с другими. Спасибо всем участникам этого сообщества за их усилия и за участие в создании этой книги.

Сотрудничающие с нами авторы помогли сделать книгу более совершенной и всеобъемлющей, и мы не смогли бы добиться такого результата без их содействия, поэтому мы выражаем огромную благодарность за помощь. Мирча Улинич (Mircea Ulinic) написал раздел SaltStack для главы об инструментальных средствах управления сетевой конфигурацией, а Джере Джулиан (Jere Julian) предоставил материал о Puppet, который мы, к сожалению, не смогли включить в текущую версию книги. Спасибо, Мирча и Джере.

Технические рецензенты весьма серьезно относились к проверке содержания книги, соблюдая баланс между точностью и правильностью излагаемого материала и легкостью восприятия этого материала читателем. Мы выражаем свою благодарность Патрику Огенстаду (Patrick Ogenstad), Ахилу Белу (Akhil Behl), Эрику Чоу (Eric Chou) и Сринивасу Макаму (Sreenivas Makam). Спасибо за то, что помогли!

Список людей, заслуживающих благодарности, был бы неполным без упоминания в нем сотрудников O'Reilly Media: Вирджиния Уилсон (Virginia Wilson) и Куртни Аллен (Courtney Allen) – редакторы, Дуайт Рэмси (Dwight Ramsey) – литературный редактор, Рэчел Монэхан (Rachel Monaghan) – корректор, Джуди МакКонвил (Judy McConville) – составитель предметного указателя, Колин Коул (Colleen Cole) – технический редактор, Рэнди Комер (Randy Comer) – дизайнер обложки и Ребекка Демаре (Rebecca Demarest) – художник-иллюстратор. Важность их помощи на всем пути книги от идеи до воплощения невозможно переоценить, и мы благодарим этих людей за их увлеченность работой и вклад в создание книги.

Глава 1

Тенденции в современной промышленной эксплуатации сетей

Вам мало знаком термин «программно определяемая сеть» (Software Defined Networking, SDN)? Или вы захвачены массовым увлечением технологией SDN в последние несколько лет? В какую бы категорию вы ни попали, не стоит беспокоиться. Эта книга проведет вас по всем основным темам и поможет начать изучение сетевой программируемости и автоматизации с момента появления SDN. В данной главе представлен обзор тенденций в современной промышленной эксплуатации сетей, при этом особое внимание уделено SDN, ее значимости и воздействию этой технологии на современный мир сетей. Начнем мы с исторического обзора внедрения SDN в основной пул технологий и окончательного формирования направлений сетевой программируемости и автоматизации.

ВОЗНИКНОВЕНИЕ ТЕХНОЛОГИИ ПРОГРАММНО ОПРЕДЕЛЯЕМОЙ СЕТИ

Если бы нужно было назвать только одного человека, который изменил всю сетевую индустрию, это был бы Мартин Касадо (Martin Casado), в настоящее время являющийся главным партнером (General Partner) и вкладчиком-инвестором (Venture Capitalist) в венчурном фонде Andreessen Horowitz. Ранее Касадо был действительным членом совета VMware, первым вице-президентом

и генеральным директором подразделения Networking and Security Business в компании VMware. Он оказал огромное воздействие на всю сетевую индустрию не только непосредственным участием в разработках (включая OpenFlow и Nicira), но и прояснением общей ситуации в отношении обязанностей, возлагаемых на крупные сети, а также острой необходимости внесения изменений в функционирование, адаптируемость и управляемость сетей. Рассмотрим эти факты более подробно.

OpenFlow

Как бы то ни было, OpenFlow послужил в качестве первого основного протокола в процессе внедрения и продвижения программно определяемой (или конфигурируемой) сети (Software Defined Network, SDN). Мартин Касадо разрабатывал протокол OpenFlow в процессе получения кандидатской степени (PhD) в Стэнфордском университете (Stanford University) под руководством Ника МакКеона (Nick McKeown). OpenFlow представляет собой протокол, который всего лишь позволяет отделить уровень управления сетевыми устройствами от уровня представления данных (см. рис. 1.1). Проще говоря, уровень управления можно интерпретировать как разум, мозг (brains) сетевого устройства, а уровень представления данных – как аппаратуру (hardware) или интегральную схему специального назначения (application-specific integrated circuits, ASIC), которая действительно выполняет перенаправление (forwarding) пакетов.



Рис. 1.1 ❖ Разделение уровня управления и уровня данных с помощью протокола OpenFlow

Работа OpenFlow в гибридном режиме

На рис. 1.1 показаны элементы сети, не имеющие уровня управления. Эта схема представляет «чистый» вариант развертывания на основе только протокола OpenFlow. Многие устройства также поддерживают работу OpenFlow в гибридном режиме, то есть протокол OpenFlow может быть развернут на заданном порту, в виртуальной локальной сети (VLAN) или даже в обычном канале перенаправления пакетов таким образом, что если в таблице OpenFlow не найдено соответствие, то используются существующие таблицы перенаправления (MAC-адреса, таблицы маршрутизации и т. п.). Такой режим работы в большей степени похож на маршрутизацию на основе правил (Policy Based Routing, PBR).

Все сказанное выше означает, что OpenFlow представляет собой протокол низкого уровня, используемый для прямого взаимодействия с таблицами аппаратных устройств (например, с информационной базой данных переадресации (Forwarding Information Base, FIB)), которые указывают сетевому устройству, как перенаправлять трафик (например, «трафик на целевой адрес 192.168.0.100 должен исходить из порта 48»).

i OpenFlow – это протокол низкого уровня, который управляет таблицами потоков (данных), то есть напрямую воздействует на перенаправление (переадресацию) пакетов. OpenFlow не предназначен для взаимодействия с атрибутами уровня управления, такими как процедура аутентификации или параметры протокола SNMP.

Поскольку таблицы OpenFlow не ограничиваются поддержкой только целевых адресов, в отличие от обычных протоколов маршрутизации, они обеспечивают бóльшую детализацию (соответствие полей в пакетах) при определении пути перенаправления. Но этот подход отличается от детализации, предлагаемой маршрутизацией на основе правил (PBR). Подобно OpenFlow в далеком будущем, PBR-маршрутизация позволяет сетевым администраторам перенаправлять трафик на основе «не совсем обычных» атрибутов, таких как адрес источника пакетов. Но при этом для поставщиков сетевого оборудования потребовалось некоторое время, чтобы обеспечить соответствующую производительность при PBR-маршрутизации трафика, поэтому окончательный результат оставался в весьма сильной зависимости от поставщиков. Появление протокола OpenFlow означало, что теперь можно достичь того же уровня детализации в принятии решений о переадресации, но при этом полностью устраняется зависимость от производителей и поставщиков оборудования. Появилась возможность расширить функциональные возможности сетевой инфраструктуры без ожидания следующей версии аппаратных устройств от производителей.

История программируемых сетей

OpenFlow не был самым первым протоколом или технологией, используемой для отделения функций управления и принятия решений от сетевых устройств. Его появлению предшествовала долгая история создания технологий и исследований, тем не менее

именно OpenFlow стал технологией, начавшей «революцию программно определяемых сетей». Предшественниками OpenFlow являются такие технологии, как Forwarding and Control Element Separation (ForCES), Active Networks, Routing Control Platform (RCP) и Path Computation Element (PCE). Чтобы более подробно ознакомиться с историей развития программно определяемых сетей и соответствующих технологий, прочтите статью «The Road to SDN: An Intellectual History of Programmable Networks» (<https://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/sdnhistory.pdf>), авторы: Джен Рексфорд (Jen Rexford), Ник Фимстер (Nick Feamster) и Элен Зегура (Ellen Zegura).

Почему именно OpenFlow?

Несмотря на важность понимания того, что представляет собой протокол OpenFlow, еще более важно понять основания и причины, по которым начались исследования и разработка первоначальных технических характеристик OpenFlow и последующее появление программно определяемых сетей.

Мартин Касадо работал на национальное правительство во время своего обучения в Стэнфордском университете. В процессе этой работы возникла необходимость организации ответных действий при атаках, угрожающих безопасности IT-систем (прежде всего IT-систем правительства США). Касадо быстро понял, что можно создать программу, которая управляет компьютерами и серверами именно так, как ему нужно. Реальные примеры использования этой программы никогда не публиковались, но это был тот тип управления конечными точками сети, который позволял предпринимать ответные действия, анализировать и в перспективе перепрограммировать хост или группу хостов в тех случаях, когда это необходимо.

В то время в реальных сетях было почти невозможно выполнить задуманное исключительно программным способом. Каждое сетевое устройство было «закрытым» (например, блокировалась установка любого стороннего программного обеспечения) и предлагало только интерфейс командной строки (command-line interface, CLI). Хотя интерфейс командной строки был и остается широко распространенным и даже предпочитаемым инструментом сетевых администраторов, Касадо ясно понял, что этот инструмент не способен обеспечить гибкость, требуемую для управления, эксплуатации и защиты сети.

В действительности за последние 20 лет способ управления сетями не изменился, за исключением добавления функциональных возможностей с помощью новых команд в интерфейс командной строки. Самым крупным изменением был переход с Telnet на SSH, послуживший основой для шутки, часто используемой SDN-компанией Big Switch Networks на своих демонстрационных слайдах, один из которых можно видеть на рис. 1.2.

Но если говорить серьезно, то технология управления сетями значительно отстала от других технологий, поэтому Касадо в конечном итоге занялся этой проблемой, чтобы изменить положение дел в течение нескольких следующих лет. Скучность средств управления часто лучше осознается при изучении других технологий. Другие технологии почти всегда предлагают более современ-

ные способы управления большим количеством устройств и для управления конфигурацией, и для сбора и анализа данных – например, менеджеры гипервизоров, контроллеры беспроводных сетей, телефонные системы IP PBX, PowerShell, инструменты DevOps – список можно продолжать бесконечно. Некоторые из перечисленных средств являются коммерческим программным обеспечением, то есть собственностью производителя, но прочие инструменты можно свободно применять для управления, эксплуатации и обеспечения гибкости сетей независимо от конкретной платформы.

PROBLEM: NETWORK AGILITY
Not Much has Changed in the Last 20 Years

1994	2014
<pre>Router> enable Router# configure terminal Router(config)# enable secret cisco Router(config)# ip route 0.0.0.0 0.0.0.0 20.2.2.3 Router(config)# interface ethernet0 Router(config-if)# ip address 10.1.1.1 255.0.0.0 Router(config-if)# no shutdown Router(config-if)# exit Router(config)# interface serial0 Router(config-if)# ip address 20.2.2.2 255.0.0.0 Router(config-if)# no shutdown Router(config-if)# exit Router(config)# router rip Router(config-router)# network 10.0.0.0 Router(config-router)# network 20.0.0.0 Router(config-router)# exit Router(config)# exit Router# copy running-config startup-config Router# disable Router></pre>	<pre>Router> enable Router# configure terminal Router(config)# enable secret cisco Router(config)# ip route 0.0.0.0 0.0.0.0 20.2.2.3 Router(config)# interface ethernet0 Router(config-if)# ip address 10.1.1.1 255.0.0.0 Router(config-if)# no shutdown Router(config-if)# exit Router(config)# interface serial0 Router(config-if)# ip address 20.2.2.2 255.0.0.0 Router(config-if)# no shutdown Router(config-if)# exit Router(config)# router rip Router(config-router)# network 10.0.0.0 Router(config-router)# network 20.0.0.0 Router(config-router)# exit Router(config)# exit Router# copy running-config startup-config Router# disable Router></pre>
Terminal Protocol: Telnet	Terminal Protocol: SSH

Рис. 1.2 ❖ Что изменилось? Telnet заменили на SSH
(источник: Big Switch Networks)

Если вновь вернуться к ситуации, когда Касадо работал на правительство, то возникает вопрос: существовала ли возможность перенаправления трафика на уровне прикладного программного обеспечения? Существовали ли в то время прикладные программные интерфейсы (API) для сетевых устройств? Существовал ли единый пункт обмена информацией для всей сети? На все эти вопросы в большинстве случаев следовал отрицательный ответ. Каким образом появилась возможность программирования сети для динамического управления перенаправлением пакетов, стратегиями и конфигурациями с такой же простой реализацией, как написание программы и выполнение ее на оконечной хост-машине?

Первоначальная спецификация OpenFlow стала результатом исследований Мартина Касадо, который искал пути решения перечисленных выше проблем. Когда ажиотаж вокруг OpenFlow прекратился, поскольку в итоге внимание сосредоточилось в большей степени на вариантах применения и практических решениях, нежели на протоколах низкого уровня, первая работа Касадо стала катализатором для переосмысления всего процесса создания, управления и эксплуатации сетей. Благодарим тебя, Мартин.

Кроме того, это значит, что если бы не было Мартина Касадо, то, возможно, эта книга никогда не была бы написана. Хотя кто знает...

Что такое программно определяемая сеть

В предыдущем разделе был кратко описан протокол OpenFlow, но что такое программно определяемая (или конфигурируемая) сеть (Software Defined Networking, SDN)? Это одно и то же или нет, а может быть, это совершенно разные вещи? Откровенно говоря, SDN – это практически то же самое, чем была облачная среда (Cloud) около десяти лет назад, до тех пор, пока мы не узнали о различных типах облака: инфраструктура как сервис (Infrastructure as a Service, IaaS), платформа как сервис (Platform as a Service, PaaS) и программное обеспечение как сервис (Software as a Service, SaaS).

Есть множество примеров и проектных решений, способствующих пониманию того, чем была облачная среда и чем она стала сейчас, но даже до того, как появились вышеперечисленные термины, могли возникать споры о том, что именно подразумевается при упоминании «облачной среды». Аналогичная ситуация и с термином «программно определяемая сеть». Существуют общедоступные определения, утверждающие, что сетевая среда как «прозрачный ящик» – это SDN или что наличие прикладного программного интерфейса (API) для сетевого устройства – это SDN. Но действительно ли такие характеристики определяют термин SDN? Разумеется, нет.

Вместо попыток определения SDN мы рассмотрим следующие технологии и направления, которые весьма часто ассоциируют с SDN и упоминают при обсуждении SDN:

- OpenFlow;
- виртуализация сетевых функций (Network Functions Virtualization, NFV);
- виртуальная коммутация;
- виртуализация сети;
- прикладные программные интерфейсы (API) устройств;
- автоматизация сети;
- аппаратная коммутация;
- сетевые фабрики центров данных;
- SD-WAN (программно определяемая глобальная сеть);
- специализированные сетевые контроллеры.

i Мы преднамеренно не даем определение программно определяемой сети (SDN) в этой книге. Несмотря на то что SDN упоминается в текущей главе, главное внимание здесь сосредоточено на основных направлениях и тенденциях, которые часто классифицируются как SDN. Это сделано для того, чтобы читатель узнал как можно больше о каждом из этих направлений.

Из всех перечисленных направлений основное внимание в книге уделено автоматизации сети, прикладным программным интерфейсам (API) и объемлющим (внешним) технологиям, которые чрезвычайно важны для понимания того, каким образом все эти компоненты объединяются в сетевых устройствах, предоставляющих программируемые интерфейсы в совокупности с современными инструментальными средствами автоматизации и управления.

OpenFlow

Протокол OpenFlow уже был представлен читателю, но есть еще несколько важных фактов, которые следует знать.

Одним из самых главных преимуществ использования протокола, подобного OpenFlow, между контроллером и сетевыми устройствами была реальная независимость провайдера (организатора сети) от программного обеспечения контроллера, иногда обозначаемого как сетевая операционная система (network operating system, NOS), и от виртуальных и физических сетевых устройств более низкого уровня. Тем не менее в действительности сетевые провайдеры, использующие OpenFlow в своих решениях (например, Big Switch Networks, HP, NEC), разработали расширения протокола OpenFlow в соответствии с развитием стандартов и из-за необходимости предоставления специфических дополнительных весьма полезных функциональных возможностей, которые базовая серийная версия OpenFlow предложить не может. Но можно предположить, что в конечном итоге все эти расширения будут включены в будущие версии, реализующие стандарт OpenFlow.

При использовании OpenFlow вам предоставляется большой уровень детализации при управлении маршрутами трафика в сети, но большая власть влечет за собой и большую ответственность. Хорошо, если у вас есть группа разработчиков. Например, корпорация Google на основе протокола OpenFlow развернула глобальную сеть В4, которая увеличила производительность почти на 100%. Для большинства других организаций использование OpenFlow или другого подобного протокола будет менее важным, чем комплексные всеобъемлющие решения, ориентированные на поддержку конкретного типа бизнеса.

i Несмотря на то что этот раздел называется OpenFlow, с точки зрения архитектуры он посвящен разделению уровня управления и уровня данных. OpenFlow – это основной протокол, используемый для выполнения данной функции.

Виртуализация сетевых функций

Виртуализация сетевых функций (Network Functions Virtualization, NFV) представляет собой не слишком сложную концепцию сетевой архитектуры. По су-

ществу, она означает развертывание функций, которые ранее обычно выполнялись аппаратурой, в виде программного обеспечения. Самыми известными примерами реализации этой концепции являются виртуальные машины, работающие как маршрутизаторы, сетевые экраны (брандмауэры), балансировщики нагрузки, IDS/IPS, VPN, защитные экраны прикладного уровня (технология WAF) и выполняющие многие другие функции/сервисы.

Появление NFV позволило ослабить казавшуюся незыблемой базовую позицию аппаратуры, стоимость которой могла достигать десятков или даже сотен тысяч долларов с сотнями и тысячами строк команд, и перейти к конфигурированию сети с помощью N компонентов программного обеспечения, то есть с помощью виртуальных устройств. Эти небольшие устройства стали намного более управляемыми, если рассматривать их как отдельные независимые сетевые устройства.

i При описанном выше подходе используются виртуальные устройства как форм-фактор для реальных устройств, поддерживающих концепцию виртуализации сетевых функций. Это всего лишь один из примеров. Развертывание сетевых функций как программного обеспечения может происходить во многих различных формах, включая встраивание в гипервизор, в виде контейнера или как приложение, работающее на x86-сервере.

Нередко встречаются случаи развертывания оборудования, потребность в котором с большой вероятностью возникнет в течение будущих трех–пяти лет, так как постепенные обновления слишком сложны и даже являются более дорогостоящими. Таким образом, эксплуатируемую основную стоимость составляет не только оборудование, оно лишь используется для сценариев типа «что, если» для вариантов роста и расширения. Развертывание решений, основанных на программном обеспечении, то есть NFV-решений, предлагает более эффективный способ масштабирования и сведения к минимуму проблемной области критических сбоев всей сети или конкретного приложения при использовании модели «оплата по мере роста» (pay-as-you-grow model). Например, вместо приобретения одного крупного аппаратного межсетевого экрана Cisco ASA вы можете постепенно развертывать виртуальные устройства Cisco ASAv и платить по мере роста. Точно с такой же легкостью можно масштабировать балансировщики нагрузки с помощью новейших технологий, предлагаемых компаниями, подобными Avi Networks.

Если концепция NFV способна предложить столь весомые преимущества, то почему не наблюдается изобилия реально внедренных в эксплуатацию решений и программных продуктов этой категории? По нескольким различным причинам. Во-первых, необходимо полное переосмысление архитектуры сети. При установке единственного монолитного сетевого экрана (как пример) все проходят через этот экран, то есть все приложения и все пользователи, а если не все, то определенная группа, известная системному администратору. В современной модели NFV, где возможно развертывание многочисленных виртуальных сетевых экранов, создается сетевой экран для каждого приложения или

пользователя в противоположность одному общему экрану, «ответственному за все». Такой подход существенно сужает проблемную область критических сбоев, ограничивая ее одним сетевым экраном или любым другим сервисным устройством, а при необходимости внесения изменений или развертывания нового приложения никоим образом не затрагиваются другие сетевые экраны для прочих приложений (или пользователей).

С другой стороны, в более привычном всем мире монолитных устройств чрезвычайно важно наличие единой панели управления стратегией защиты, обычно в форме интерфейса командной строки (CLI) или графического пользовательского интерфейса (GUI). Возможно, такой подход значительно расширяет потенциальную проблемную область критических сбоев, но предлагает администраторам четко определенную стратегию управления, поскольку управление требуется для единственного устройства. При наличии группы или отдела поддержки таких устройств вполне естественным становится выбор монолитно-ориентированного подхода. Сегодня это действительность, но есть надежда, что через некоторое время с появлением более усовершенствованных инструментов, оказывающих помощь в эксплуатации и управлении решениями, основанными на программном обеспечении, в промышленном масштабе мы увидим более широкое практическое применение этого типа технологии. В мире современных автоматизированных сетевых операций и автоматизированного управления сетями все меньшее значение имеет выбор сетевой архитектуры с точки зрения продуктивности выполнения функций, поскольку имеется возможность намного более эффективного управления как одним устройством, так и большим количеством устройств.

Наряду с вопросами управления другим фактором, влияющим на ситуацию, является то, что многие производители уделяют недостаточно внимания продажам своих версий виртуальных устройств. Мы не утверждаем, что у производителей отсутствуют версии с виртуализацией, но обычно эти версии не являются предпочтительными у многих известных производителей сетевого оборудования. Если поставщик начал свою деятельность по продаже сетевого оборудования в течение нескольких последних лет, то можно заметить его резкий переход к модели, основанной на программном обеспечении, с учетом перспективы продаж и получения прибыли.

Как и во многих технологичных отраслях, одним из главных преимуществ концепции NFV также является ее гибкость и адаптируемость. Исключение аппаратных устройств уменьшает время внедрения новых сервисов, поскольку не требуется время на установку и монтирование оборудования и кабелей, на настройку и подключение к существующей сетевой среде. При использовании программной методики время подключения зависит только от времени развертывания новой виртуальной машины в сетевой среде, а присущие такому подходу преимущества можно получать многократно при клонировании и резервировании виртуальных устройств для будущего тестирования, например в рабочих средах восстановления после катастроф (disaster recovery, DR).

Наконец, при развертывании NFV исчезает необходимость маршрутизации трафика через специализированное физическое устройство с целью создания требуемого сервиса.

Виртуальная коммутация

На сегодняшний день самыми известными на рынке средствами виртуальной коммутации являются VMware standard switch (VSS), VMware distributed switch (VDS), Cisco Nexus 1000V, Cisco Application Virtual Switch (AVS) и виртуальный коммутатор с открытыми исходными кодами Open vSwitch (OVS).

Эти коммутаторы весьма часто упоминаются в спорах о сущности SDN, но в действительности они представляют собой программные коммутаторы, которые размещены в ядре гипервизора, предоставляя возможность установления соединений в локальной сети между виртуальными машинами (а в последнее время и между контейнерами). Программные коммутаторы предоставляют такие функции, как распознавание MAC-адреса, и такие возможности, как объединение каналов связи, SPAN и sFlow, аналогичные функциям их физических конкурентов, выполняемым в течение многих лет. Несмотря на то что эти виртуальные коммутаторы часто применяются в более крупных решениях SDN и виртуализации сети, они представляют собой лишь коммутаторы, реализованные в форме программного обеспечения. Хотя сами по себе виртуальные коммутаторы не являются законченным решением, они чрезвычайно важны, так как способствуют прогрессу всей отрасли. Виртуальные коммутаторы создали новый уровень доступа или новое направление в деятельности центров данных. На границах сети уже не применяется физический стоечный (top-of-rack, TOR) коммутатор в аппаратном исполнении с ограниченной гибкостью (с точки зрения развития возможностей/функциональности). С тех пор, как границы сетей стали «программными», благодаря использованию виртуальных коммутаторов появилась возможность более быстрого создания новых сетевых функций, реализуемых в программном обеспечении, следовательно, существенно упростилось распространение стратегий управления по сети. Например, стратегия управления защитой может быть развернута на порту виртуального коммутатора, ближайшего к реальному конечному узлу сети. Коммутатор может быть виртуальной машиной или контейнером, предназначенным для дальнейшего расширения стратегии защиты для всей сети.

Виртуализация сети

Решения, классифицируемые как виртуализация сети, уже стали синонимами SDN-решений. В рамках тематики этого раздела виртуализация сети обозначает исключительно программные решения, основанные на оверлейном протоколе. Самыми известными решениями из этой категории являются VMware NSX, Nuage Virtual Service Platform (VSP) и Juniper Contrail.

Главная характеристика этих решений – оверлейный протокол, такой как Virtual eXtensible LAN (VxLAN), используемый для создания соединений между виртуальными коммутаторами на основе гипервизора. Этот тип соединений

и методика туннелирования (tunneling) обеспечивает совместимость на уровне канала передачи (Layer 2) между виртуальными машинами, существующими на различных независимых физических хостах физической сети, при этом подразумевается, что физическая сеть может быть сетью уровня 2 (Layer 2), уровня 3 (Layer 3) или комбинацией обоих уровней. В результате создается отделенная от физической сети виртуальная сеть, которая предоставляет свободу выбора и гибкость.

i Следует отметить, что термин оверлейная сеть (overlay network) часто используется вместе с термином андерлейная (нижележащая) сеть (underlay network). Поэтому следует уточнить, что андерлейная означает нижележащая физическая сеть, созданная с помощью физических кабелей. Оверлейная сеть формируется с использованием решения виртуализации сети, которое динамически создает туннели (tunnels) между виртуальными коммутаторами в пределах центра данных. И снова отметим, что это следует рассматривать в контексте решения виртуализации сети на основе программного обеспечения. Также отметим, что многие решения на аппаратной основе в настоящее время развертываются с помощью VxLAN в качестве оверлейного протокола для создания туннелей уровня 2 (Layer 2, каналный) между стоечными (top-of-rack) устройствами в пределах центра данных уровня 3 (Layer 3, сетевой).

Несмотря на то что оверлейный протокол является лишь элементом реализации решения виртуализации сети, эти решения представляют собой гораздо большее, нежели простой набор виртуальных коммутаторов, объединяемых оверлеями. Обычно это полные, всеобъемлющие решения, предлагающие функции защиты, балансировки нагрузки и обратную совместимость при взаимодействии с физической сетью, имеющей единственный пункт управления (например, контроллер). Часто такие решения также обеспечивают объединение с сервисами лучших компаний в сфере сетевых услуг уровнями 4–7 (Layer 4–7), предлагая на выбор ряд технологий, которые можно развернуть на платформах виртуализации сети.

Кроме того, гибкость (адаптируемость) достигается благодаря наличию центральной управляющей платформы, которая используется для динамического конфигурирования каждого виртуального коммутатора и необходимых сервисных компонентов. Напомним, что застой в развитии эксплуатационных характеристик сетей произошел из-за вездесущего преобладания интерфейса командной строки (CLI), предпочитаемого всеми производителями в реальном рыночном мире. При виртуализации сети нет необходимости вручную конфигурировать все виртуальные коммутаторы, так как любое решение упрощает этот процесс, предоставляя централизованный интерфейс (GUI, CLI), а также прикладной программный интерфейс (API), с помощью которого все изменения можно реализовать программно.

Прикладные программные интерфейсы сетевых устройств

В течение нескольких последних лет производители начали понимать, что одного лишь стандартного интерфейса командной строки уже недостаточно

и его использование ограничивает функциональность. Если вы когда-либо работали с любым языком программирования или скриптовым языком, то, вероятнее всего, сможете понять эту проблему. В любом случае, мы более подробно рассмотрим эту тему в главе 7.

Главным проблемным пунктом является то, что скрипты для старых и современных сетевых устройств, использующих только интерфейс командной строки, не возвращают структурированные данные. То есть предполагается, что данные передаются из устройства в скрипт в простейшем текстовом формате (например, вывод команды `show version`), затем требуется написание отдельного скрипта для синтаксического разбора этого текста, чтобы извлечь необходимые атрибуты, такие как время непрерывной работы или версию операционной системы. Даже при незначительных изменениях вывода команд `show` скрипты могут стать неработоспособными из-за неверных правил синтаксического разбора. Хотя такой подход применялся и продолжает применяться всеми администраторами, с технической точки зрения автоматизация стала возможной уже давно, но только сейчас производители постепенно переходят на сетевые устройства, управляемые прикладными программными интерфейсами (API).

Предоставление API исключает необходимость синтаксического разбора неструктурированного текста, так как сетевое устройство возвращает структурированные данные, существенно сокращая время, затрачиваемое на написание скрипта. Вместо парсинга «сырого» текста в поисках времени непрерывной работы или любого другого атрибута возвращается объект, предоставляющий именно ту информацию, которая необходима в данном конкретном случае. При этом не только сокращается время написания скрипта и снижается «входной» уровень требований для сетевых инженеров (и прочих непрограммистов), но также предоставляется более четко определенный интерфейс, с помощью которого профессиональные разработчики программного обеспечения могут быстро разрабатывать и тестировать код почти так же, как они используют API для несетевых устройств. «Тестирование кода» может означать проверку (испытания) новых топологий, сертификацию новых сетевых функциональных возможностей, валидацию конкретных сетевых конфигураций и многое другое. Все эти операции сегодня выполняются вручную, на них затрачивается очень много времени, при этом весьма высока вероятность возникновения ошибок.

Один из первых широко распространенных API-интерфейсов в сетевой сфере представила компания Arista Networks. Ее интерфейс называется eAPI и представляет собой прикладной программный интерфейс на основе протокола HTTP, который использует данные в JSON-кодировке. Разумеется, API на основе протокола HTTP и использование формата JSON будут подробно рассматриваться начиная с главы 5. После Arista компания Cisco представила свои API-интерфейсы, такие как Nexus NX-API и NETCONF/RESTCONF, на конкретных платформах, и появились производители и поставщики, такие как

Juniper, получившие в свое полное распоряжение расширяемый и адаптируемый интерфейс NETCONF, но это не привлекло особого внимания. Следует отметить, что в те времена почти каждый производитель/поставщик предлагал собственный тип API.

Более подробно эта тема будет рассматриваться в главе 7.

Автоматизация сети

По мере развития прикладных программных интерфейсов в сетевой отрасли продолжают появляться все более интересные варианты извлечения преимуществ из их использования. В ближайшем будущем автоматизация сети является главным потенциальным средством получения преимуществ программируемых интерфейсов при использовании современных сетевых устройств, предоставляющих API.

В более широком смысле автоматизация сети – это не только автоматизация процедур конфигурирования сетевых устройств. Разумеется, это самое распространенное представление об автоматизации сети, но использование API и программируемых интерфейсов обеспечивает гораздо большие потенциальные возможности автоматизации, нежели простое манипулирование параметрами конфигурации.

Переход к использованию API делает простым и ясным доступ ко всем данным, скрытым в сетевых устройствах. Здесь имеются в виду данные уровня информационных потоков, таблицы маршрутизации, информационные базы данных переадресации (FIB – Forwarding Information Base), статистические данные интерфейсов, таблицы MAC-адресов, таблицы виртуальных локальных сетей (VLAN), серийные номера устройств – этот список можно продолжать бесконечно. Использование современных методик автоматизации, которые, в свою очередь, применяют API, может оказать оперативную помощь в выполнении повседневных операций управления сетью, сбора данных и автоматизированной диагностики. Кроме того, поскольку современные API позволяют извлекать структурированные данные, администраторы получают возможность выводить и анализировать именно те наборы данных, которые необходимы, и даже объединять выходные данные от нескольких различных команд show, значительно сокращая время на отладку и устранение проблем в сети. Вместо установления соединений с N-маршрутизаторами, работающими по протоколу граничного шлюза (BGP – Border Gateway Protocol), с целью проверки правильности конфигурации или устранения возникшей проблемы можно воспользоваться методиками автоматизации для упрощения этого процесса.

Кроме того, применение методик автоматизации в конечном итоге позволяет создать более предсказуемую и единообразную сеть в целом. Это можно наблюдать при автоматизации создания файлов конфигурации, при автоматизации создания виртуальной локальной сети (VLAN) и/или при автоматизации процесса устранения проблем и неисправностей. Это упрощает понимание обобщенного процесса всеми пользователями, поддерживающими конкретную сетевую среду, в отличие от ситуации, в которой каждый сетевой адми-

нистратор должен формировать свою собственную индивидуальную практическую методику работы.

Различные типы автоматизации сети подробно рассматриваются в главе 2.

Аппаратная коммутация

Аппаратную коммутацию (*bare-metal switching*) тоже часто приравнивают к SDN, но в действительности это ошибочное представление. Ранее уже было отмечено, что в этой главе даются краткие описания различных технологических направлений, которые воспринимаются как SDN, следовательно, и этой теме нужно уделить немного внимания. Если вернуться в 2014 год (и даже немного раньше), то можно обнаружить, что для обозначения аппаратной коммутации в то время использовался термин «коммутация с помощью прозрачного ящика» (*white-box switching*) или *commodity switching*. С тех пор термин изменился, и не без серьезных на то оснований.

Прежде чем рассматривать замену «прозрачного ящика» (*white-box*) на «голую» аппаратуру (*bare-metal*), необходимо понять, что это означает на самом верхнем уровне с учетом существенных изменений в представлениях о том, что представляют собой сетевые устройства. В последние 20 лет сетевые устройства всегда приобретались как физические устройства, точнее, как аппаратные устройства, сетевая операционная система и инструменты/приложения, которые можно было использовать в сетевой операционной системе. Все эти компоненты приобретались у одного поставщика/производителя.

В соответствии с концепцией прозрачного ящика и «чисто» аппаратных сетевых устройств такое устройство по большей части похоже на x86-сервер (см. рис. 1.3). Это позволяет пользователю отделить друг от друга все требуемые компоненты, предоставляя возможность купить аппаратуру у одного производителя, операционную систему у другого производителя, а затем подбирать и загружать инструменты/приложения от различных поставщиков или даже воспользоваться программным обеспечением с открытыми исходными кодами.

Коммутация по методике прозрачного ящика стала весьма популярной в период повышенного интереса к OpenFlow, поскольку целью этой методики являлось превращение аппаратуры в товар массового потребления и сосредоточение «интеллекта» сети в OpenFlow-контроллере, который сейчас называют SDN-контроллером. В 2013 году корпорация Google объявила о создании собственных коммутаторов под управлением OpenFlow. Тогда это объявление стало предметом многочисленных дискуссий между профессионалами, но в действительности далеко не каждого конечного пользователя можно сравнить с Google и не каждый пользователь будет создавать собственную программную или аппаратную платформу.

Вместе с этими событиями наблюдалось появление нескольких компаний, сосредоточивших свои усилия на предоставлении решений на основе коммутации по методике прозрачного ящика, например Big Switch Networks, Cumulus Networks и Pica8. Каждая такая компания предлагает решения исключительно

на основе программного обеспечения, но при этом остается необходимость в аппаратуре, на которой должно работать программное обеспечение, чтобы предоставить полноценное завершённое решение. Первоначальным источником этих аппаратных платформ «прозрачного ящика» были ODM-компании¹, такие как Quanta, Super Micro и Accton. Даже если вы работали в сетевой отрасли, вряд ли вам что-либо известно об этих компаниях.

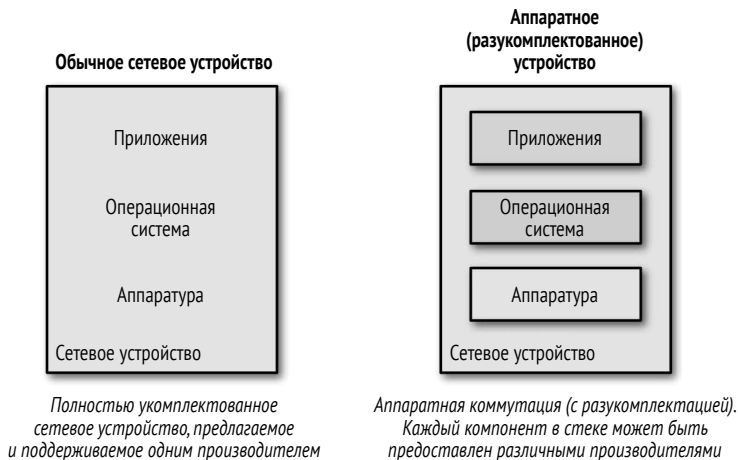


Рис. 1.3 ❖ Общая схема обычного стека коммутации и стека аппаратной коммутации

До тех пор, пока компании Cumulus и Big Switch не объявили о сотрудничестве с крупными корпорациями, в том числе с HP и Dell, не менялось и название этого направления – «прозрачного ящика» на «аппаратную коммутацию», но теперь крупные известные производители поддерживали операционные системы от других компаний, таких как Big Switch и Cumulus Networks, на своих аппаратных платформах.

Возможно, до сих пор остается недопонимание того факта, что аппаратная коммутация с технической точки зрения не является SDN, поскольку производители, подобные Big Switch, действуют в обоих направлениях. На возникающие в связи с этим вопросы можно дать простой ответ. Если контроллер объединен с программным решением, использующим протокол типа OpenFlow (но не обязательно OpenFlow), а обмен данными между сетевыми устройствами организован с помощью программного обеспечения, то все это в совокупности очень похоже на программно определяемую сеть (SDN). Именно так действует компания Big Switch – загружает программное обеспечение на «голую» аппаратуру (прозрачный ящик), затем инициализирует работу агента OpenFlow, обеспечивающего обмен данными с контроллером как компонентом этого решения.

¹ ODM – Original Design Manufacturers.

С другой стороны, компания Cumulus Networks предлагает специализированный дистрибутив Linux, ориентированный на работу с сетевыми коммутаторами. Этот дистрибутив, или операционная система, поддерживает реализации обычных стандартных протоколов, таких как LLDP, OSPF и BGP, без каких-либо конкретных требований к контроллеру, то есть обеспечивает сопоставимость и совместимость с сетевыми архитектурами, которые не основаны на SDN.

Приведенное выше описание должно стать свидетельством того, что Cumulus является компанией, создающей сетевые операционные системы и обеспечивающей работу своего программного обеспечения на чисто аппаратных коммутаторах, в то время как Big Switch – компания, создающая SDN на аппаратной основе, где требуется использование SDN-контроллера этой компании, но при этом Big Switch поддерживает инфраструктуру аппаратной коммутации других производителей.

Если подвести краткий итог, то аппаратную коммутацию (коммутатор как прозрачный ящик) можно приблизительно определить как разукomплектацию (разделение на отдельные компоненты) и предоставление возможности приобретения сетевой аппаратуры у одного производителя, а программного обеспечения у другого, право выбора остается за пользователем. В этом случае администраторы получают полную свободу при выборе проектных решений, архитектур и программного обеспечения без необходимости замены аппаратуры; иногда требуется заменить только основную операционную систему.

Сетевые фабрики центров данных

Возникала ли у вас ситуация, когда невозможно обеспечить простое взаимодействие между различными сетевыми устройствами, даже если все они работают по стандартным протоколам, таким как Spanning Tree или OSPF? Если эта проблема вам знакома, то вы не одиноки. Представьте себе сеть центра данных с компактным ядром и отдельными коммутаторами в каждой аппаратной стойке. А теперь подумайте, какие действия необходимо выполнить, когда наступает время обновления.

Существует много способов обновления сетей, подобных вышеупомянутой, но что, если надо обновить только стоечные (top-of-rack, TOR) коммутаторы и при исследовании текущего рынка TOR-коммутаторов был выбран новый производитель или новая платформа? Это вполне обычная ситуация, она возникает достаточно часто. Сам процесс прост – обеспечение взаимодействия между новыми коммутаторами и существующим ядром (разумеется, мы предполагаем, что в ядре имеются доступные порты) и правильное конфигурирование магистрального канала связи по стандарту 802.1Q, если это соединение уровня 2 (канальный уровень), или конфигурирование предпочитаемого протокола маршрутизации, если это соединение уровня 3 (сетевой уровень).

Здесь начинают действовать сетевые фабрики центров данных (data center network fabrics). Именно они призваны полностью изменить представление о работе сетей центров данных.

Сетевые фабрики центров данных ориентированы на изменение образа мышления сетевых операторов: от управления отдельными устройствами

поочередно к управлению всей системой как единым целым. Если вернуться к описанному в начале раздела сценарию, то здесь вряд ли появится возможность замены на TOR-коммутатор от другого производителя, так как коммутатор является всего лишь отдельным компонентом сети центра данных. Но когда сеть развертывается и управляется как единая система, необходимо воспринимать ее как единую систему. Это означает, что процесс обновления должен представлять собой переход от системы к системе, от фабрики к фабрике. В таком контексте фабрики могут полностью заменяться во время процедур обновления, но не допускается замена отдельных компонентов внутри фабрики, по крайней мере, в большинстве случаев. Такой подход возможен, если конкретный производитель/поставщик предоставляет подробный план (маршрут) перехода или обновления и только при использовании аппаратной коммутации (то есть заменяется только аппаратура). Примерами сетевых фабрик центров данных являются Cisco Application Centric Infrastructure (ACI), Big Switch Big Cloud Fabric (BCF) или фабрика и гиперконвергентная сеть (hyper-converged network) компании Plexxi (<https://www.plexxi.com/>).

В дополнение к интерпретации сети как единой системы сетевые фабрики центров данных также обладают следующими общими характеристиками:

- единый интерфейс для управления и конфигурирования всей фабрики, включая управление стратегиями;
- распределенные по всей фабрике шлюзы, определяемые по умолчанию;
- возможности определения многих различных маршрутов;
- использование одной из форм SDN-контроллера для управления системой.

SD-WAN

В последние два года одним из самых интенсивно разрабатываемых направлений SDN стала методика программно определяемой глобальной сети (Software Defined Wide Area Networking, SD-WAN). За последние годы выросло количество компаний, всерьез занявшихся решением проблем, связанных с глобальными сетями (WAN), в их числе Viptela (совсем недавно поглощенная корпорацией Cisco), CloudGenix, VeloCloud, Cisco IWAN, Glue Networks и Silverpeak.

В сфере глобальных сетей не наблюдалось значительных технологических прорывов со времени перехода с протокола ретрансляции кадров Frame Relay на механизм коммутации MPLS. С появлением широкополосных каналов связи и интернета затраты становятся сопоставимыми с затратами на аналогичные частные линии связи, со временем развитие продолжается с применением VPN-туннелей, связывающих сайты, в итоге все это закладывается в основу следующей важной вехи – глобальной сети (WAN).

Часто применяемые проектные решения для удаленных офисов обычно включают частную линию связи (с универсальным многопротокольным механизмом передачи данных – multiprotocol label switching, MPLS) и/или открытое интернет-соединение. При наличии обоих вариантов интернет обычно используется только как резервный вариант, главным образом для входящего («гостевого») трафика или для данных общего пользования, передаваемых

через VPN в корпоративную систему, тогда как MPLS-канал используется для приложений, требующих минимальных задержек, таких как голосовая связь или канал видеосвязи. Когда трафик начинает разделяться между каналами связи, увеличивается сложность конфигурирования протокола маршрутизации, а также ограничивается детализация при определении маршрутов к целевому адресу. Адрес источника, приложение и производительность (пропускная способность) сети в режиме реального времени, как правило, не принимаются во внимание в решениях по определению наилучшего маршрута.

Общая архитектура программно определяемой глобальной сети (SD-WAN), используемая во многих современных решениях, похожа на решение по виртуализации сети, используемое в центре данных, тем, что здесь также применяется оверлейный протокол для взаимодействия граничных устройств SD-WAN. Поскольку задействован оверлейный протокол, в таком решении не имеет значения, какая именно физическая среда передачи данных более низкого уровня используется в данном конкретном случае, что обеспечивает функционирование SD-WAN через интернет или поверх частной глобальной сети (WAN). Эти решения нередко работают на основе двух и более каналов связи интернета на вспомогательных сайтах с полным зашифрованием трафика с использованием IPsec. Кроме того, многие из этих решений постоянно измеряют производительность каждого канала связи, чтобы при перегрузках быстро переключаться между каналами, обеспечивая работу важных приложений даже в самые сложные периоды. Так как здесь прикладной уровень зримо участвует в процессе, администраторы также могут выбирать приложения, для которых должен определяться конкретный маршрут. Подобного рода функциональные возможности не часто обеспечиваются в архитектурах глобальных сетей, которые полагаются исключительно на маршрутизацию по предопределенному целевому пункту с использованием обычных протоколов маршрутизации, таких как OSPF и BGP.

С точки зрения архитектуры решения SD-WAN от упоминаемых выше производителей, таких как Cisco, Viptela и CloudGenix, также обычно предлагают некоторую форму автоматической установки и конфигурирования (zero-touch provisioning, ZTP) и централизованного управления с помощью портала на стороне клиента или в облачной среде в виде SaaS-приложения, существенно упрощающего управление и эксплуатацию развивающейся глобальной сети.

Полезный побочный эффект от использования технологии SD-WAN проявляется в том, что эта технология предлагает конечным пользователям большую свободу выбора, так как в глобальной сети, как и в интернете, можно использовать любое устройство передачи данных и любой тип соединения. При этом упрощается конфигурирование и снижается сложность основных магистральных сетей, что, в свою очередь, позволяет упростить их внутреннюю организацию и архитектуру и, возможно, сократить расходы на их поддержку. Если взглянуть чуть дальше с технической точки зрения, то можно утверждать, что все конструкции логических сетей, таких как Virtual Routing and Forwarding (VRF), должны управляться через пользовательский интерфейс (user interface, UI) на

платформе контроллера, который предоставляет производитель SD-WAN. При этом опять же исчезает необходимость ожидания в течение нескольких недель ответной реакции на затребованные пользователем изменения.

Специализированные сетевые контроллеры

Возможно, вы уже заметили, что некоторые из описанных выше направлений и тенденций взаимосвязаны в той или иной степени. Это один из моментов, сложных для понимания любой новой технологии и любого нового направления развития техники в последние несколько лет.

Например, широко применяемые платформы виртуализации сетей используют контроллер как средство нескольких вариантов решений, которые также попадают в категории сетевых фабрик центров данных, SD-WAN и аппаратной коммутации. Вы в замешательстве? Возможно, у вас возникает вопрос: почему сеть на основе контроллеров сама по себе не выделена в отдельную категорию? В действительности чаще всего это лишь характеристика и механизм реализации современных решений, но не все описанные выше направления в полной мере охватывают функциональные возможности, которые могут предоставить контроллеры с технической точки зрения.

Например, широко известный SDN-контроллер OpenDaylight (ODL), показанный на рис. 1.4, как и многие другие контроллеры, представляет собой платформу, а не рыночный продукт. Это платформы, которые могут предлагать специализированные приложения, например виртуализацию сети, но, кроме

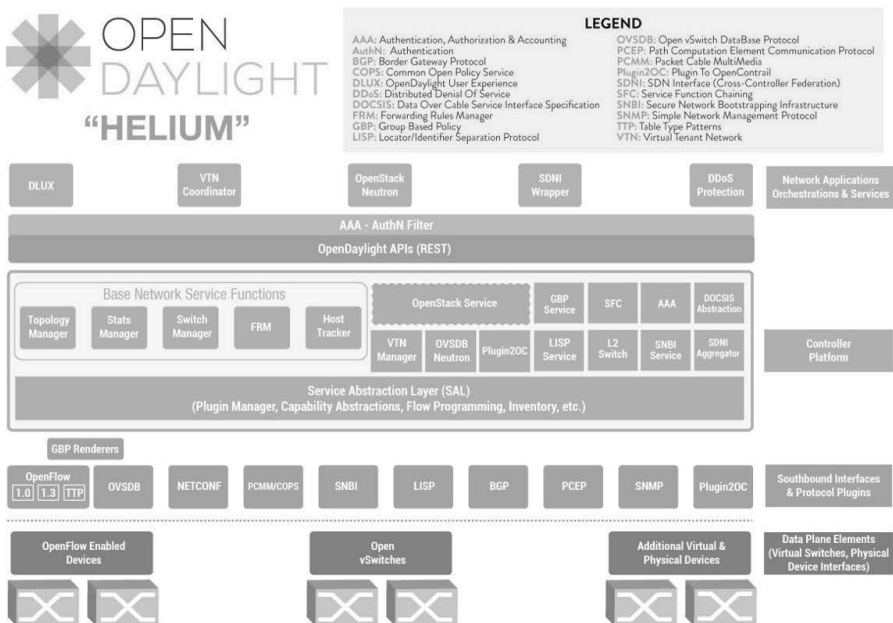


Рис. 1.4 ❖ Архитектура контроллера (платформы) OpenDaylight

того, эти платформы используются для мониторинга и наблюдения за состоянием сетей, для компоновки и укрупнения сетей и для выполнения многих других функций в сочетании с приложениями, работающими поверх платформы контроллера. Именно поэтому очень важно понять, что могут предложить контроллеры сверх того, что предлагают более привычные приложения, такие как фабрики, средства виртуализации сети и SD-WAN.

РЕЗЮМЕ

Вы получили общее представление о направлениях и технологиях, которые чаще всего относят к категории программно определяемых сетей, указывающих путь к повышению эффективности операций, выполняемых в сети, с помощью автоматизации и реализации программируемости сети. За последние несколько лет были созданы десятки SDN-стартапов, в них инвестированы миллионы в форме венчурных капиталов, и миллиарды потрачены на приобретение (поглощение) этих компаний. Может показаться невероятным, но если заглянуть чуть дальше в будущее, то все это делалось и делается с одной общей целью – реализовать программные принципы и технологии, чтобы предложить пользователям бóльшую мощь, управляемость, гибкость и свободу выбора применяемой технологии с одновременным повышением эффективности эксплуатации сетей.

В главе 2 будет рассматриваться автоматизация сети, будут более подробно описаны различные типы автоматизации, некоторые основные протоколы и прикладные программные интерфейсы, а также успехи в развитии автоматизации сетей за последние несколько лет.