Ответы на экзаменационные билеты по курсу “Програмно-конфигурируемые сети” (2016)

*Составители: Кочетков Павел,*

*Сычёва Евгения,*

*Новосёлов Алексей,*

*Хахалин Анатолий*

[1. Введение в парадигму SDN, OpenFlow. OpenFlow контроллер. Отличие SDN от OpenFlow.](#h.vyvmjcbpcp7l)

[2. Проблематика программируемости OpenFlow приложений: композиция приложении и разрешение конфликтов.](#h.3zd0nsx91a22)

[Композиция приложении и разрешение конфликтов](#h.34xgs54zgxdq)

[3. Проблематика программируемости OpenFlow приложений: система автоматической генерации правил.](#h.kr42tnd9ftq9)

[4. Применение SDN в корпоративном сегменте.](#h.1m83ur80hv4s)

[5. Применение SDN в ЦОД.](#h.3oik6ldzar6m)

[6. Применение SDN в Телеком.](#h.9acw8vwromz8)

[7. Применение SDN в домашних сетях.](#h.wgh1s5q9ksiw)

[8. Организация OpenFlow коммутатора. Основные трудности. Гибридные и чистые OpenFlow коммутаторы.](#h.rtjb5v1kgto4)

[9. Организация OpenFlow коммутатора. Аппаратный OpenFlow коммутатор.](#h.wani07f0mmn2)

[10. Организация OpenFlow коммутатора. Программный OpenFlow коммутатор.](#h.cn23ihr8swz5)

[11. Альтернативные подходы к управлению SDN коммутаторами: OFDPA.](#h.ibcd0yi20t3)

[12. Альтернативные подходы к управлению SDN коммутаторами: NetConf.](#h.x7hfpidtkfgk)

[13. Протоколонезависимые SDN коммутаторы: POF.](#h.t6wojzadedlg)

[14. Протоколонезависимые SDN коммутаторы: P4.](#h.jkdjd563w489)

[15. SDN в беспроводных сетях: CapWap.](#h.fm6epsdppncx)

[16. SDN в беспроводных сетях: SDR.](#h.eotf2mwrcmk3)

[17. Примеры практического применения SDN&NFV в России.](#h.tf5aclk5g9yv)

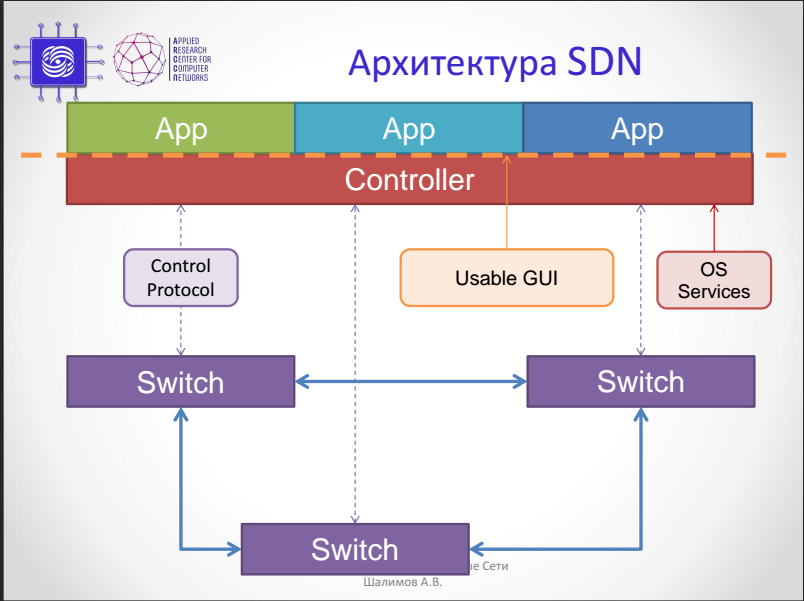
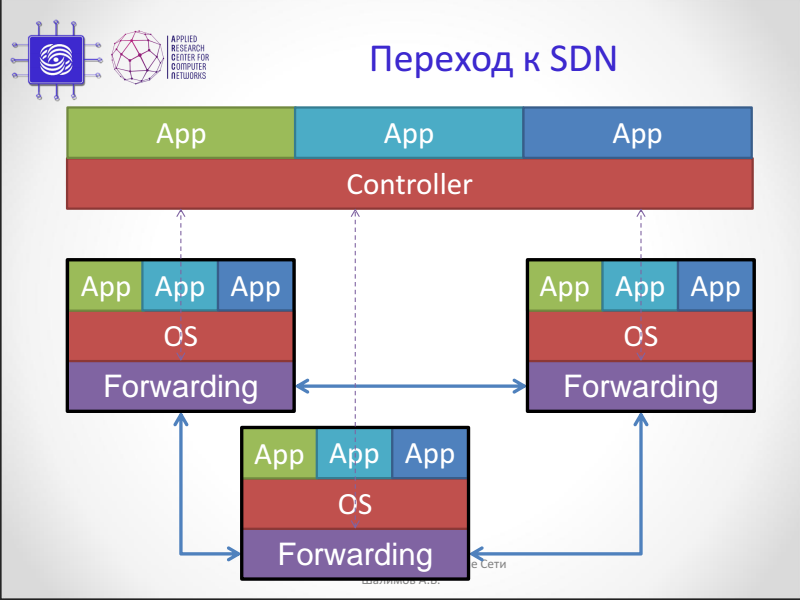
# 

# 1. Введение в парадигму SDN, OpenFlow. OpenFlow контроллер. Отличие SDN от OpenFlow.

## Введение в парадигму SDN, OpenFlow.

*Программно-Конфигурируемые Сети* (Software Defined Networking/SDN) – это разделение плоскости передачи и управления данными, позволяющее осуществлять программное управление плоскостью передачи, которое может быть физически или логически отделено от аппаратных коммутаторов и маршрутизаторов.

1. Отделить управление сетевым оборудованием от управления передачей данных за счет создания специального программного обеспечения.
2. Перейти от управления отдельными экземплярами сетевого оборудования к управлению сетью в целом.
3. Создать интеллектуальный, программно-управляемый интерфейс между сетевыми приложениями и транспортной сетью.



Достоинства такого подхода:

•Удешевление оборудования (CAPEX)

•Облегчение управления сетью (OPEX)

•Программируемость, открытость, инновации

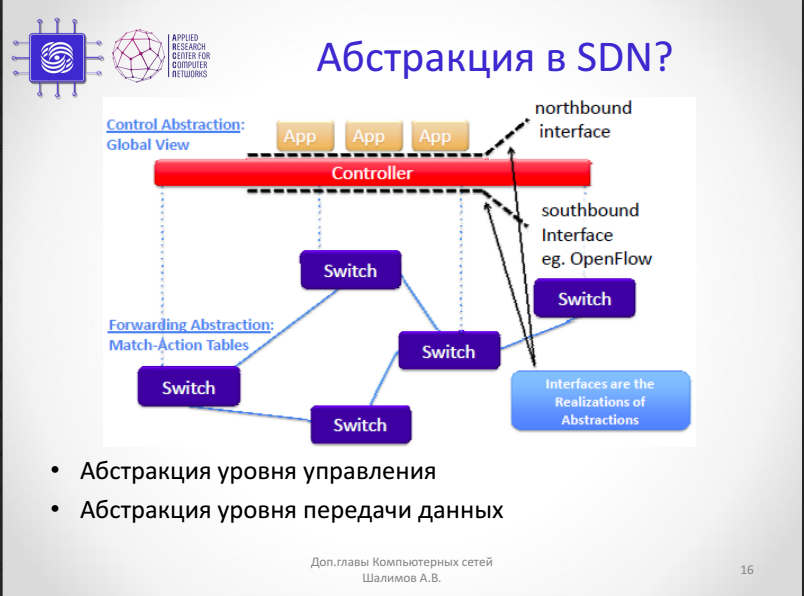
Примеры применения:

Уменьшение энергопотребления в ЦОД

–Отключение неиспользуемых коммутаторов и каналов на основе собранной информации о сети

–ElasticTree (Stanford): сокращение энергопотребления до 60%

–Применение в Google



В SDN контроллер имеет два интерфейса – для общения с приложениями (northbound interface) и для общения со свитчами (southbound interface) (строго говоря, концепция предполагает ещё присутствие east-west interface для взаимодействия нескольких экземпляров контроллера). Реализация этих интерфейсов есть абстракция на уровне

управления и на уровне передачи данных соответственно.

Для приложений это значит, что они имеют всю информацию о состоянии сети, которая им необходима, для свитчей абстракция заключается в match-action правилах.

## OpenFlow контроллер.

Чтобы коммутатор и контроллер друг друга понимали (а также могли быть отчуждаемы) необходим стандарт – OpenFlow, который задаёт формат общения, и в частности, специфицирует, какие команды может отдавать контроллер коммутатору (а т.к. контроллер подразумевает, что коммутатор выполняет его команды, то это задаёт требования к коммутатору, к его возможностям)

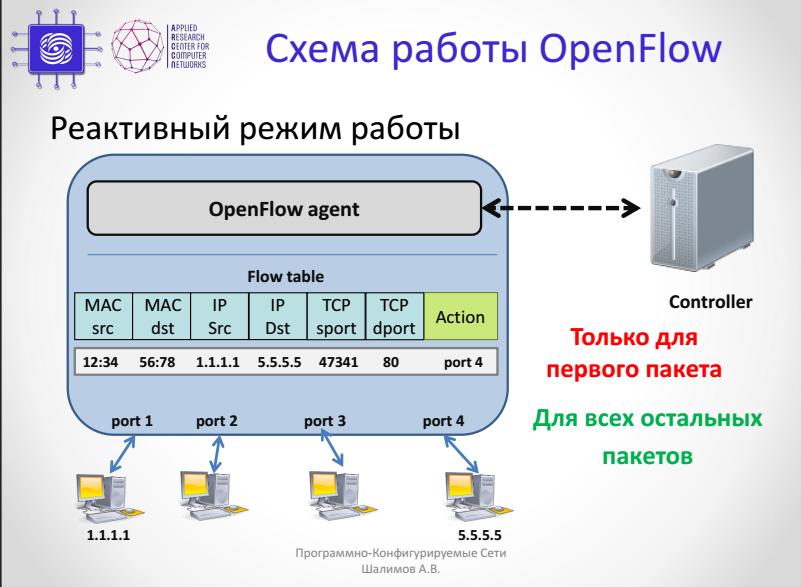
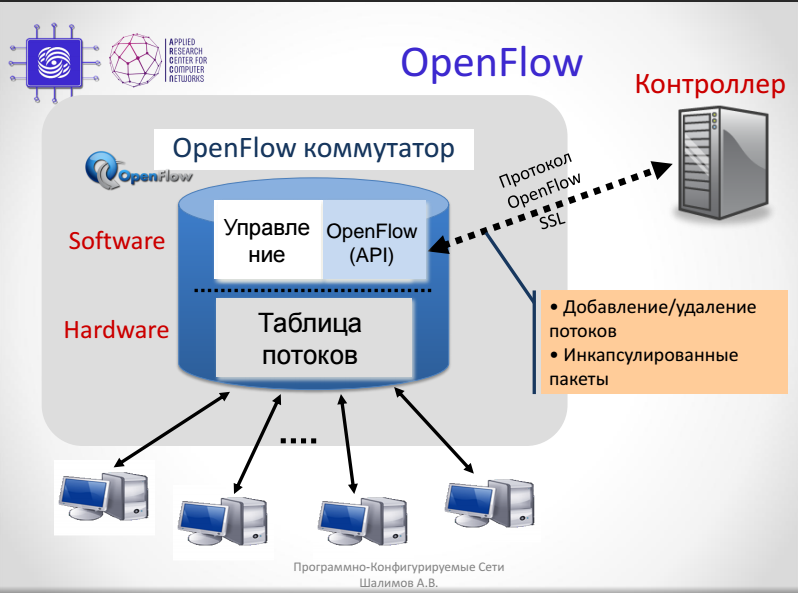


Рис. *OpenFlow контроллер*

Основная идея работы коммутатора и контроллера – на коммутаторе есть (судя по картинке аппаратная) таблица потоков, в которой хранятся правила, в соответствии с которыми коммутатор обрабатывает пакеты(можно заменять поля заголовка, отправить/сбросить пакет), если приходит пакет, который не подходит ни под одно

правило, то на контроллер отправляется packet in, а контроллер уже решает, что делать с пакетом, и отправляет на коммутатор packet out, коммутатор разбирает, что ему ответил контроллер и добавляет (или не добавляет, если так сказал контроллер) новое правило.

OpenFlow контроллер - это

* Программа, TCP/IP сервер, ожидающий подключения коммутаторов.
* Отвечает за обеспечение взаимодействие приложения-коммутатор.
* Предоставляет важные сервисы (например, построение топологии, мониторинг хостов)
* API сетевой ОС или контроллер предоставляет возможность создавать приложения на основе **централизованной модели программирования.**

Поддерживается три типа сообщений:

* Сообщения контроллер-коммутатор

– Конфигурирование коммутатора

– Управление и контроль состояния

– Управление таблицами потоков

– Features, Configuration, Modify-State (**flow-mod**), Read-State (multipart request), **Packet-out,** Barrier, Role-Request

* Симметричные сообщения

–Отправка в обоих направлениях

–Обнаружение проблем соединения контроллера с коммутатором

–Hello, Echo

* Ассиметричные сообщения

–Отправка от коммутатора к контроллеру

–Объявляют об изменении состояния сети, состояния коммутаторов

–**Packet-in**, flow-removed, port-status, error



## Отличие SDN от OpenFlow

SDN != OpenFlow, т.к. OpenFlow – это протокол общения, описывающий интерфейс контроллера и коммутатора. А SDN – это концепция, подразумевающая существование некоторого контроллера, приложений под него, соответствующих коммутаторов, ...

– SDN – подход к разделению уровня данных и уровня управления

– OpenFlow – одна из реализаций. Другие, XMPP, SNMP, overlay.

# 2. Проблематика программируемости OpenFlow приложений: композиция приложении и разрешение конфликтов.

Повышение производительности:

Самые ресурсоемкие задачи:

• Взаимодействие с OpenFlow коммутаторами:

– использование многопоточности,

– учет загрузки нитей и перебалансировка

• Получение OpenFlow пакетов из канала:

– чтение пакетов из памяти сетевой карты, минуя сетевой стек OS Linux,

– переключение контекста,

– виртуальные адреса

Пример (In-kernel контроллер) - контроллер был реализован в ядре ОС Linux:

• Супер-производительный

– нет переключений контекста при сетевом взаимодействии

– меньше времени на работу с виртуальной памятью

• Но очень сложно разрабатывать свои приложения

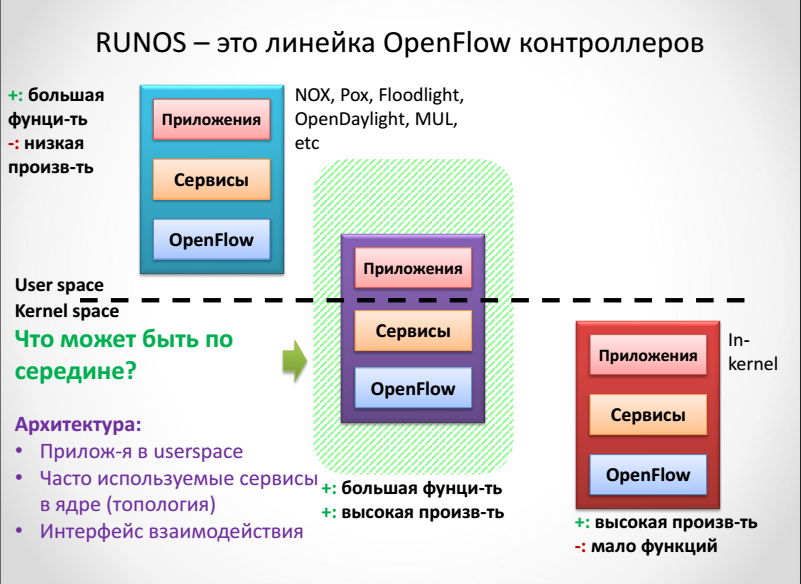
– Низкоуровневый язык программирования

– Ограниченное число библиотек и средств отладки

– Высокий риск “положить” всю систему

• Производительность равна 30M fps

• Задержка 45us



# ВариантыИсполненияRunos.png

В итоге имеем, что если мы делаем контроллер полностью в userspace, то имеем большую функциональность, но работает все медленно. Если полностью перенести контроллер в kernel-space, то писать приложения для него тяжело, риск все сломать, но зато все быстро работает.

Компромиссное решение – разместить наиболее используемые сервисы (например, определение топологии) и работу с пакетами в kernelspace, а приложения перевести в userspace, предоставив интерфейс взаимодействия kernel и user частей.

## Проблемматика программируемости

Программируемость

* На языке контроллера [быстро]
* На любом языке через REST интерфейс [медленно]
* Специальные языки программирования с другой абстракцией (например, Pyretic, Maple)

*Проблематика NorthBound API*

* NorthBound API – интерфейс между контроллером и приложениями
* Программирование с OpenFlow не простая задача!

– Сложно выполнять независимый задачи (routing, access control)

– Низкоуровневая абстракция

– Нужно помнить о правилах на коммутаторах

– Порядок установки правил на коммутаторах неизвестен

* Переносимость приложений между контроллерами

# **Композиция приложении и разрешение конфликтов**

В качестве возможной проблемы можно привести такой пример:

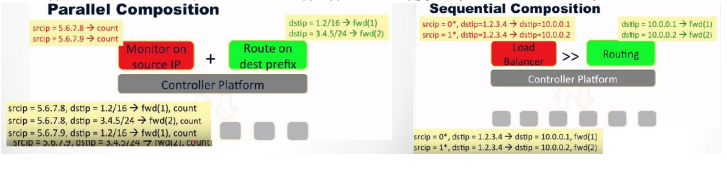
Есть два приложения, работающие независимо. Одно создает правило, что какой-то поток надо направить на порт 1, а другое – что этот же поток надо направить на порт 5. Заранее неизвестно, в каком порядке расположатся правила в коммутаторе, а пакеты будут идти только по первому правилу, поэтому работать будет только одно приложения. Хотя можно было просто сделать отправку потока на два порта.

В качестве возможного решения подобных проблем придумали композицию приложений.

Два типа композиции (на примере, Pyretic)

– *Параллельная* – выполняет оба действия одновременно (forwarding и couting)

– *Последовательная композиция* – выполняет одно действие за другим (firewall, затем switch)



Как мы с П. Поняли это.

Параллельная композиция позволяет на наборе пакетов обрабатывать “параллельно” (судя по всему предполагается, что порядок действий не важен) несколько действий (свое для каждого приложения, которые заданы на контроллере). Например, параллельная работа двух приложений: для сбора статистики (Monitor) и собственно перенаправление пакета (Route on dest prefix) - одно другому не мешает, в любом порядке можно выполнять.

Последовательная композиция позволяет использовать последовательно несколько приложений (например, балансировщик нагрузки (Load Balancer) и затем маршрутизацию (Routing)). Здесь по-любому сначала балансировщик, а потом дальнейшая маршрутизация (как на картинке). При использовании последовательной композиции сложные приложения могут быть построены из комбинации разных приложений.

На совести администратора верно выбирать или настраивать композицию приложений. В Runos была последовательная композиция приложений - мы сами определяли, какое приложение за каким будет выполняться.

А теперь как это на коммутаторе на уровне правил вFloating Table работает - непонятно.

# 3. Проблематика программируемости OpenFlow приложений: система автоматической генерации правил.

Часть 1 - См. Вопрос 2

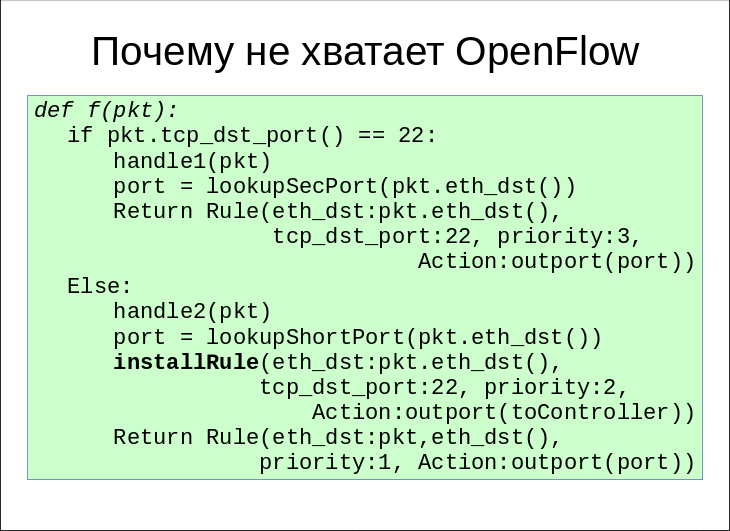
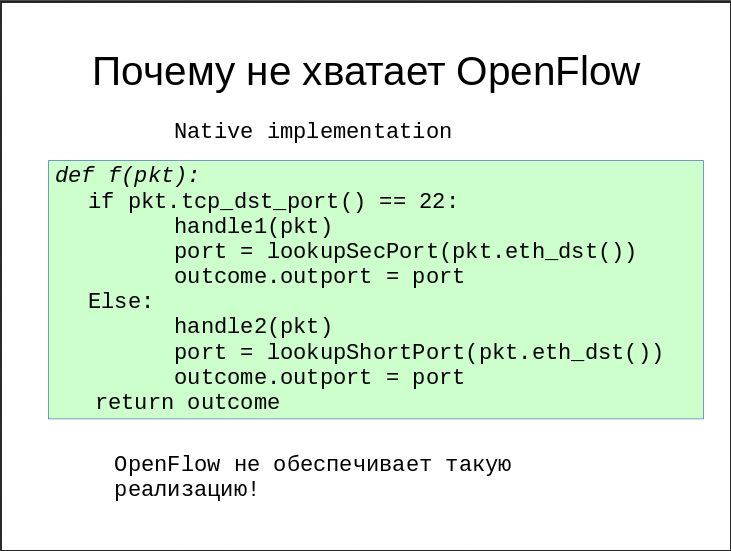
## Cистема автоматической генерации правил.

Пример проблемы

• Безопасность: TCP потоки на 22 порт идут безопасным

• Остальные потоки идут кратчайшим маршрутом

• При этом надо запоминать, где какие конечные узлы находятся



Решение:

1. Приоритеты. Понаставить везде костыльно приоритеты в коде в приложении на контроллере.

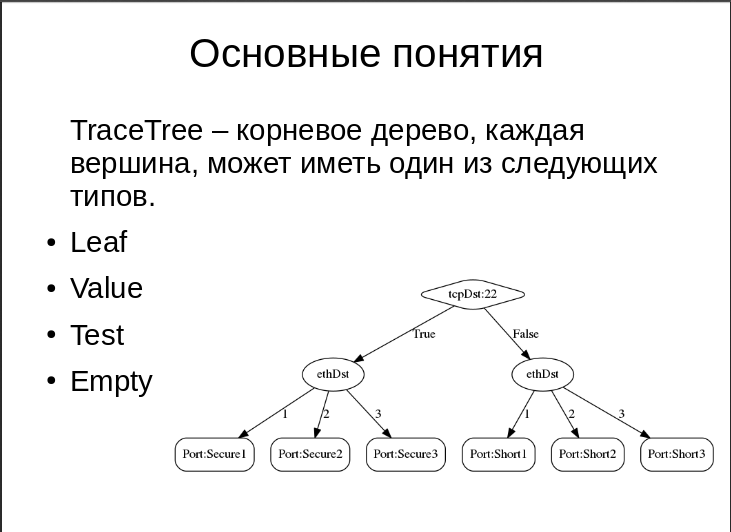


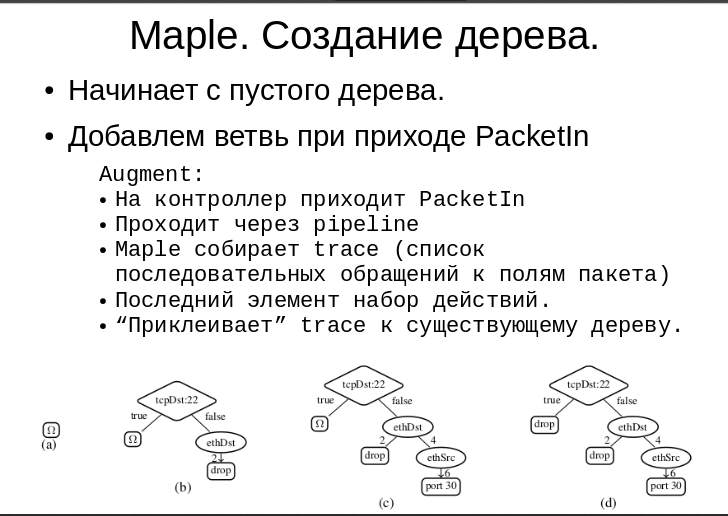
Рис X. Код генерации правил в FT для коммутатора на контроллере.

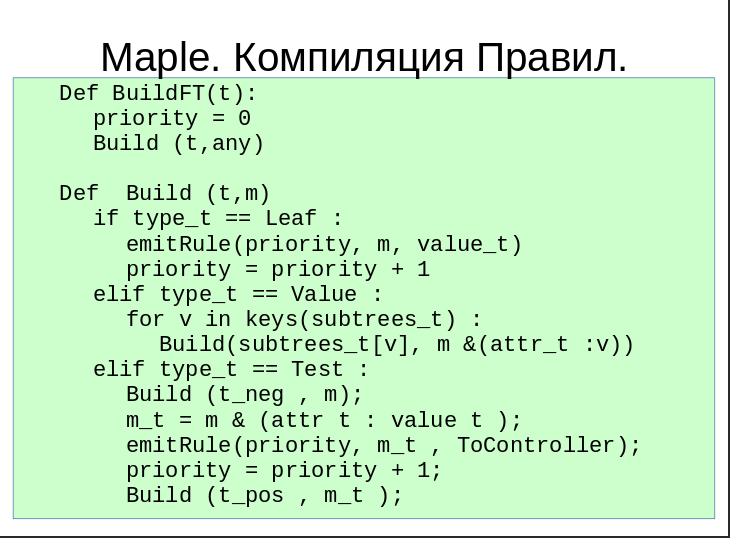
а по коду этому уже сформирмируются записи с правилами в floating таблицы на коммутаторах. Почему так костыльно? Потому что коммутатор не понимает правила типа: “c НЕ 22 порта”, поэтому на каждый случай правило.

1. Но то же самое проще получить как -Trace Tree - Построение дерева и генерация правил по дереву будет произведена автоматически с помощью Maple <http://www.cs.bu.edu/fac/byers/courses/EC/F15/Posters/example3.pdf>

Далее пример реализации кода на Maple, написать на котором проще, чем на OpenFlow, как на рис. X







Maple удобен, но не совершенен. Алгоритм Maple по построению правил можно оптимизировать, одна из работ студента 3 курса этому и была посвещена.

Оптимизации:

• По числу правил

• По приоритетам

• Генератор правил

• Добавление потоков

# 4. Применение SDN в корпоративном сегменте.

• Современные компания имеют сложную сетевую инфраструктуру:

– Большое количество сетевых элементов

– Разветвленная топология

– Набор различных политик маршрутизации и безопасности

• Сетевые администраторы отвечают за поддержания работы сетевой инфраструктуры:

– Сетевые инженеры руками переводят высокоуровневые политики в низкоуровневые команды

– Ручная настройка всех сетевых устройств

– Ограниченный инструментарий по управлению сетевыми устройствами

– Переучивание под каждого вендора

• Существующие системы управления (Примеры: Cisco Prime, HP OpenView, IBM Tivoli, OpenNMS)

- Предназначены для мониторинга состояния: топология, характеристики каналов, загрузка каналов и задержка.

- Основы на протоколе SNMP.

- Конфигурация оборудования по-прежнему происходит в ручном режиме.

Цель:

1) Сделать сеть управляемой без ручного доступа к оборудованию.

2) Повысить уровень абстракции управления сетью.

Что подталкивает корпорации к ПКС решениям

1. Увеличение важности приложений/сервисов
2. Увеличение количества приложений/сервисов
3. Автоматизация управления в сети
4. Увеличение скорости внедрения новых приложений и сервисов
5. Виртуализация



Примеры применения ПКС в корпоративных сетях:

• Сетевая виртуализация.

• Тонкое управление движением трафика в сети в зависимости от требований приложений.

• Реализация политик качества обслуживания.

• Более гибкое, тщательно контролируемое подключение пользовательских устройств к сети.

• Оптимизация WIFI роуминга в корпоративной сети.

• Оптимизация движения широковещательного трафика в сети.

• Гибкое обеспечение политик безопасности.

• Централизованное управление большим количеством устройств разных производителей.

• Балансировка нагрузки, быстрое перестроение в случае сбоя.

• Гибкое выборочное зеркалирование трафика в случае необходимости.

• Детальный мониторинг состояния сети и внедрение новых приложений имеющих возможность влиять на работу сети.

;

# 5. Применение SDN в ЦОД.

ЦОД/Облака

• Повышение утилизации оборудования и каналов

• Мониторинг и оптимизация потоков

• Виртуализация сети пользователей

• Балансировка нагрузки

• Обеспечение качества доступа

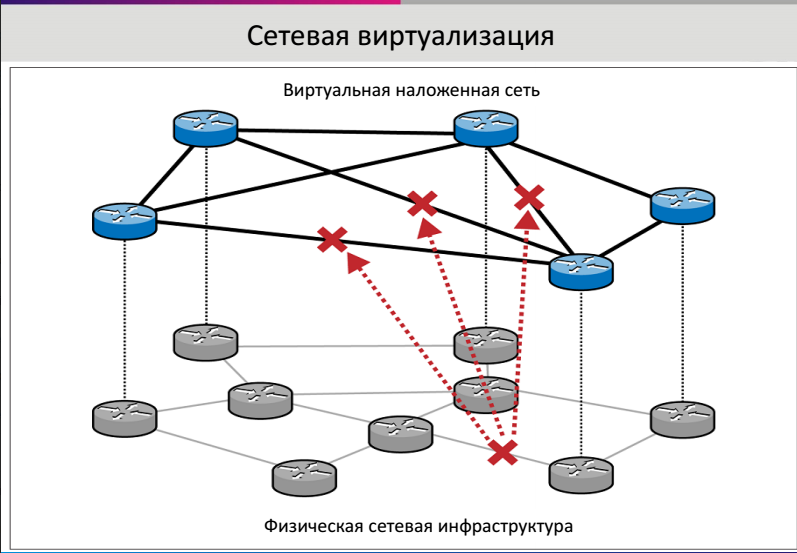
Как правило есть два SDN:

– Без OpenFlow (так в OpenStack)

ТОЛЬКО виртуальные каналы

Туннели, таблицы, новые VM, политики

– С OpenFlow для управления физическими устройствами



Сетевая виртуализация бывает:

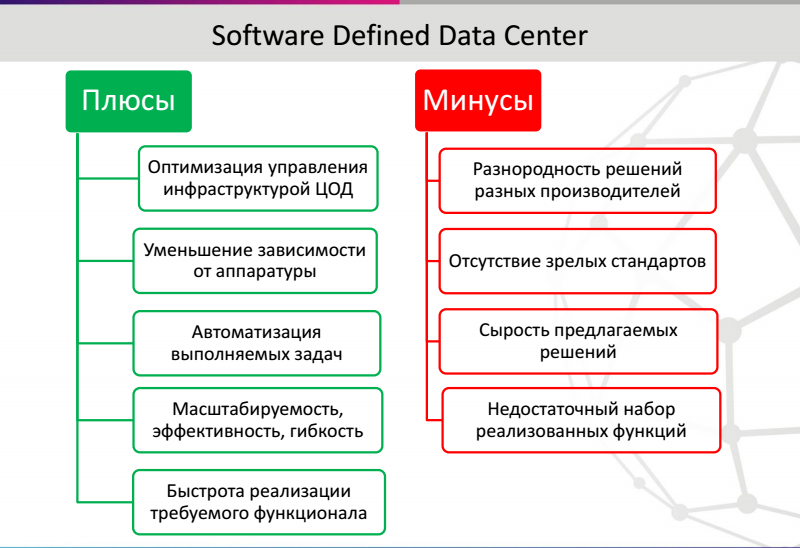
• GRE (32 bit GRE key);

• VxLAN (24 bit VxLAN Network Identifier);

• NVGRE (24 bit Virtual Subnet ID);

• Vmware/Nicira (64 bit Context ID);

• MPLS (20 bit Label)



# 6. Применение SDN в Телеком.

1. Интеллектуальный Traffic Engineering:

• Выбор оптимального пути

• Реакция на отказ канала

• Резервирование пропускной способности

Как применить все это на практике?

– Greenfield? !!! что это значит?

– Проблемы интеграции с традиционной сетью

• Нужно подыгрывать протоколам традиционной сети, т.е. правильно отвечать на запросы.

• Чем меньше стыков с традиционной сетью, тем лучше.

– Проблема интеграции с существующими системами управления

Варианты приложений ПКС контроллера

Ручная конфигурация:

• Определение пути для потока вручную.

• Мониторинг состояния транков.

Работа с маршрутной информацией:

• ПКС сегмент представлен в виде узла либо в виде линка.

• Взаимодействие с внешними источниками маршрутов:

• используя пограничное устройство;

• используя ПКС контроллер.

Расчет оптимального пути:

• Построение карты сети с использованием SNMP, RMON.

• Построение путей с учетом требований приложений и имеющихся сетевых ресурсов.

• Создание виртуальных частных сетей.

• Зеркалирование необходимых потоков.

• Балансировка нагрузки.

ПКС и глобальные сети

Технологии ПКС удовлетворят следующие требования:

•Автоматическое подключение и конфигурирование новых сайтов и линков.

•Динамическая приоритизация трафика разных типов.

•Передача данных между удаленными ЦОД.

•Гибкий защищенный удаленный доступ к ресурсам.

•Улучшенная безопасность WAN линков.

•Возможность приложений влиять на WAN связи через API контроллера.

•Мониторинг передаваемого трафика в реальном времени.

Интеллектуальная WAN среда обеспечит для распределенного ЦОД:

•Предсказуемую производительность.

•Автоматическое восстановление.

•Мониторинг загрузки, отслеживание работы приложений.

Тонкие моменты работы ПКС в WAN:

•Организация управления устройствами через неоднородную среду, публичные сети и пр.

•Безопасность трафика управления, передаваемого через рабочую сеть.

•Совместимость решений разных производителей.

# 7. Применение SDN в домашних сетях.

“SDN for Home Networks”. Цели:

Обеспечить изоляцию трафика в рамках логического уровня.

Обеспечить изоляцию полосы пропускания.

Обеспечить независимое управление логическими уровнями.

Обеспечить возможность гибкого конфигурирования индивидуального поведения логического уровня.

Пример использования ПКС в домашней сети

Оценка использования трафика различными пользователями и приложениями.

Распределение квот трафика, в том числе с учетом времени суток.

Обмен неиспользованными квотами между пользователями.

Решение следующих проблем:

• Ассоциация потоков с устройствами, а не с пользователями.

• Обеспечение различного набора политик для контроля квот пользователей, приложений и устройств.

• Обеспечение доступа к истории и текущей статистике использования трафика.

Варианты использования домашних ПКС сетей (SDHN):

Разделение домашних и гостевых пользователей.

Умные сети электроснабжения (Smart Grid).

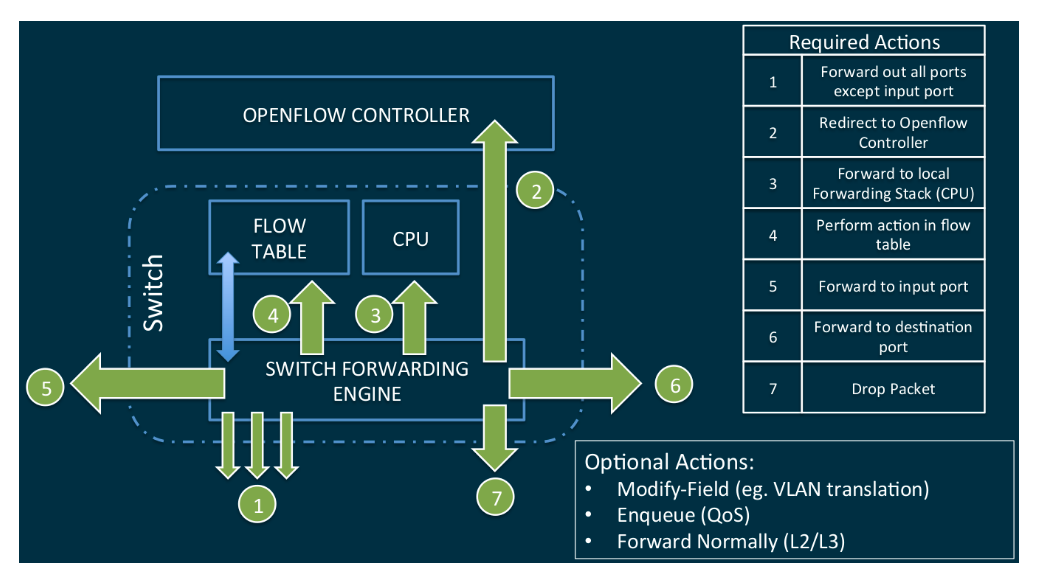
Подключение к нескольким провайдерам.

Обмен видео контентом и проведение конференций.

Использование различных технологий доступа.

Управление полосой пропускания.

# 8. Организация OpenFlow коммутатора. Основные трудности. Гибридные и чистые OpenFlow коммутаторы.



Факты:

* коммутаторы могут работать в проактивном и реактивном режиме,
* коммутатор может иметь закрытую архитектуру, но обеспечивать открытый интерфейс для управления своей таблицей коммутации.
* коммутатор может быть сравнительно простым устройством, надежным и дешевым

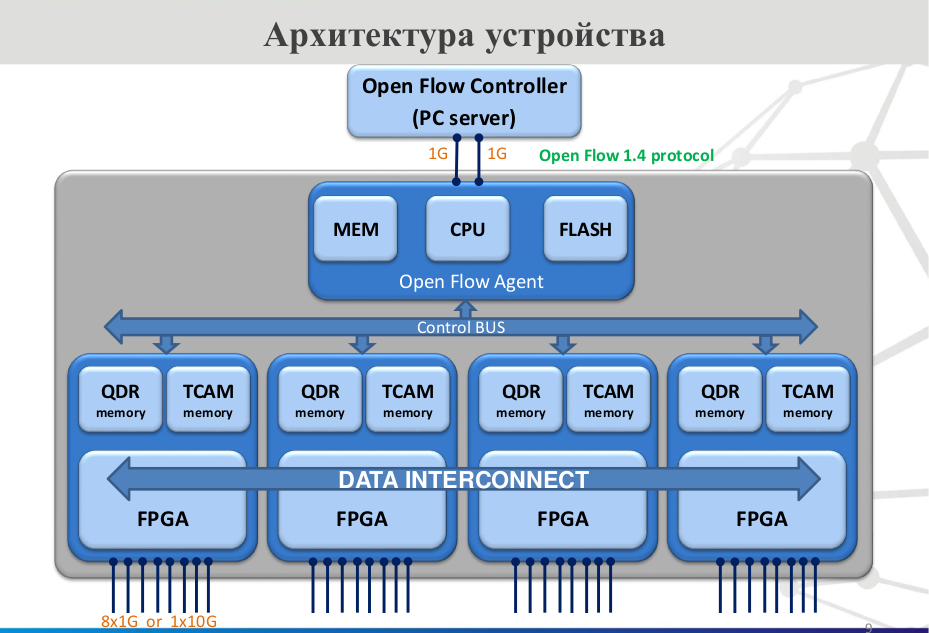
Трудности

1. Новый протокол (пр. OpenFlow), а точнее огромное количество учитываемых по умолчанию полей в таблицах маршрутизации выдвигает повышенные требования к объемам памяти на коммутаторах.
2. TCAM (Ternary Content Addressable Memory) — это единственный адекватный по скорости поиска информации способ хранения данных, от которого невозможно никуда деться, если мы хотим заниматься коммутацией и маршрутизацией на скоростях интерфейса. И стоит TCAM достаточно дорого. Поэтому даже коммутаторы на мощных относительно современных ASIC'ах порой могут поддерживать OpenFlow исключительно формально, обеспечивая менее тысячи OF-записей в TCAM.
3. можно было бы организовать многоуровневое хранение записей, сделав в TCAM буфер для самых актуальных записей и сложив в оперативную память все остальное, но для этого необходимо весьма ощутимо модифицировать платформу коммутаторов, большинство существующих моделей на такое просто не способно.
4. На разных сетевых устройствах по разному реализуется конвеер (pipeline). Отсюда вытекает проблема взаимозаменяемости. При написании приложений необходимо знать, что “под” контроллером.

Таким образом выделим 2 основных проблемы: проблема **организации памяти,** проблема **взаимозаменяемости. Память** - какую, скольки и в каких случаях использовать. **Взаимозаменяемость** - зависимость приложений от коммутаторов.

**Гибридные коммутаторы** могут функционировать как SDN устройства, так и как традиционные сетевые устройства. То есть внутри себя они располагают традиционным стеком по обработке пакетов. Такие коммутаторы могут работать в двух режимах. Помимо этого у гибридных коммутаторов существует специальный виртуальный порт NORMAL, на который может быть послан пакет для обработки традиционным способом.

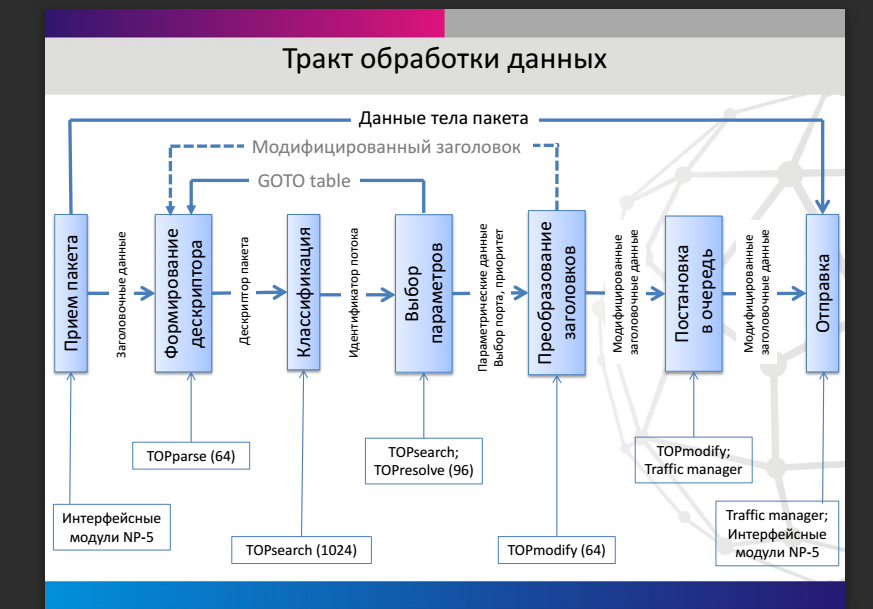
**Чистые OpenFlow коммутаторы** такой возможностью не обладают. И могут обрабатывать пакеты только за счет средств протокола OpenFlow (flow match+action) - таблиц потоков, груповых таблиц и пр.



# 9. Организация OpenFlow коммутатора. Аппаратный OpenFlow коммутатор.

Аппаратные реализации могут работать гораздо быстрее, чем их программные аналоги, таким образом, более применимы в производстве требовательных сред, например, в центрах обработки данных и core сетях.

В аппаратных коммутаторах тракт более сложные нежели в программных. Например, тракт в высокопроизводительных сетевых процессорах NP:



Пример: EZ chip

# 10. Организация OpenFlow коммутатора. Программный OpenFlow коммутатор.

Внедрение программных OpenFlow коммутаторов является самым простым способом создания SDN устройства, поскольку таблицы потоков, записи потоков и match полей, являются общими структурами данных программного обеспечения, такими как отсортированные массивы и хэш-таблицы. Следовательно, более вероятно, что два программных SDN устройства, изготовленные различными группами разработчиков будет вести себя одинаково, чем при разработки двух различных аппаратных реализаций. С другой стороны, реализации в программном обеспечении будет медленнее и менее эффективны. Следовательно, для сетевых устройств, которые должны работать на высоких скоростях, например, 10 Гбит, 40 Гбит и 100 Гбит, только аппаратные реализации возможны.

В тоже время, реализация аппаратного устройства SDN будет поддерживать только сравнительно ограниченное число таблиц потоков. В программном же коммутаторе количество записей может быть на несколько порядков больше. В программном коммутаторе можно реализовать более сложные действия.

Пример: OVS

# 11. Альтернативные подходы к управлению SDN коммутаторами: OFDPA.

### OF-DPA огромный шаг Broadcom (самый популярный производитель сетевых чипсетов, 99.98% всего интернет трафика) на встречу SDN/OpenFlow

– Описывает pipeline чипсета в терминах OpenFlow

### OF-DPA 2.0

*OF-DPA* - Broadcom OpenFlow Data Plane Abstraction

**Набор программных компонент для чипсетов по трансляции правил OpenFlow в абстракции и конвейер Broadcom чипсетов.**

* SDK, OF-DPA Driver, OF-DPA OpenFlow Agent

**OpenFlow 1.3.4**

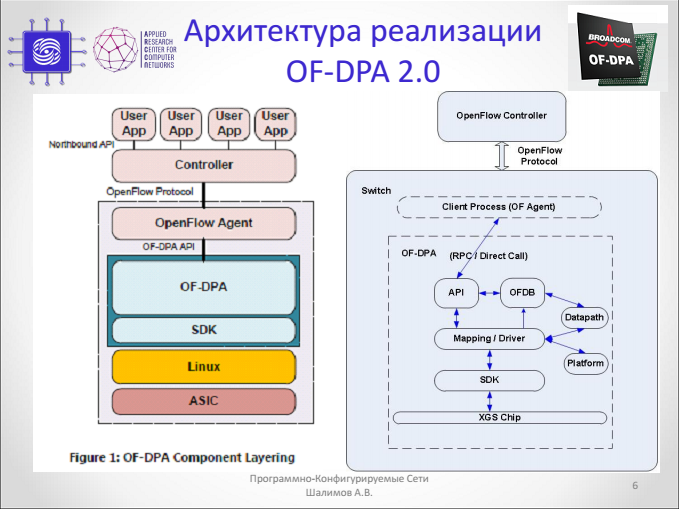
**Полная поддержка всех features от чипсета**: L2/L3 bridging, VXLAN, NVGRE, ACL (drop, rewrite), QoS, queues и т.д.

**Что описывает?**

* Какие таблицы есть, какие поля есть, какие действия, в каких таблицах возможны
* Experimental значения, чтобы разрешить специфичекую функциональность

**Что получаем?**

* Возможность программировать существующие устройства (после перепрошивки, да)
* Высокая скорость системы



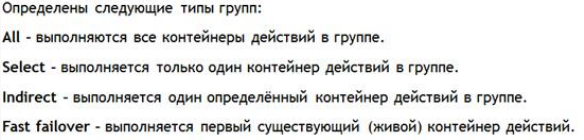
У таблиц и правил появляется **семантика**

Есть **несколько типов правил**:

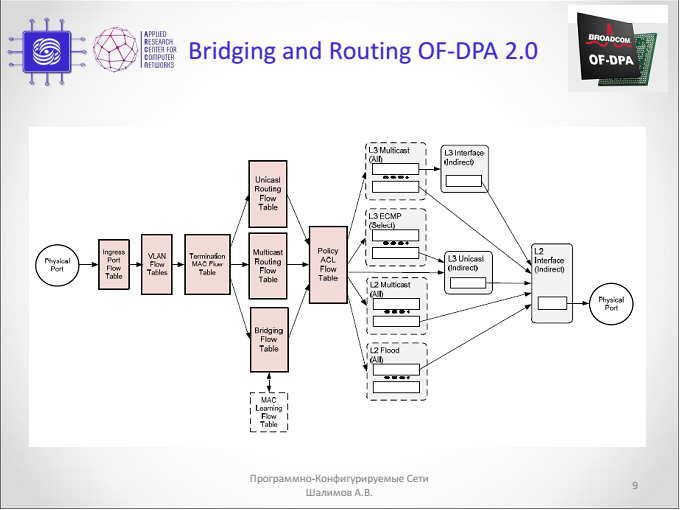
* Стандартные - мы
* Built-in - зашиты
* Automatic – добавляются автоматически

**Ограничения на групповые таблицы**

* L3 ECMP, как SELECT
* L2/L3 broadcast, как ALL



Можно **конфигурировать Switch и на самом OF-DPA SDK**. Т.е. самому специфицировать Pipeline без использования SDN/OpenFlow.



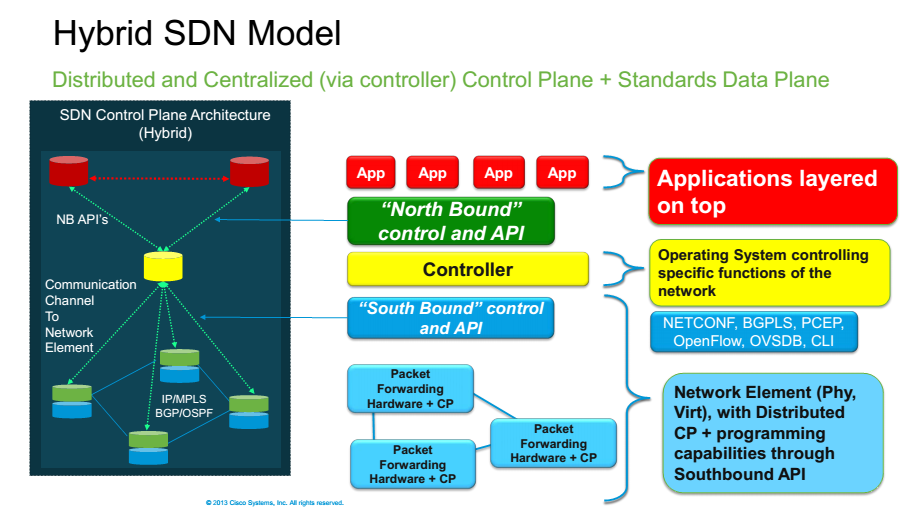
# 12. Альтернативные подходы к управлению SDN коммутаторами: NetConf.

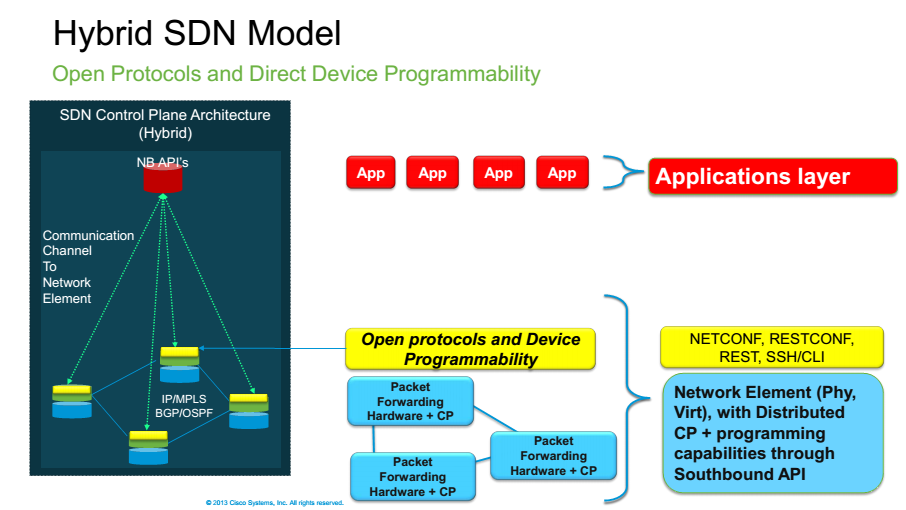
***Netconf* – network configuration protocol** - is an *IETF* (Internet Engineering Task Force) configuration management protocol

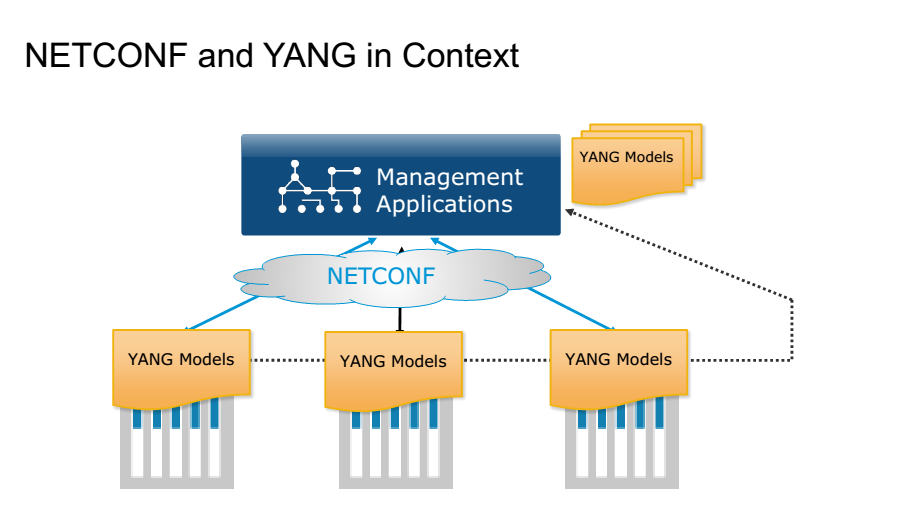
* Описывает управление конфигурацией сетевых устройств - not only configuration access, but operational state data
* На основе XML (Juniper)
* Fill the void between SNMP (monitoring) and CLI (geared towards humans)

***Yang* – Yet Another Next Generation**

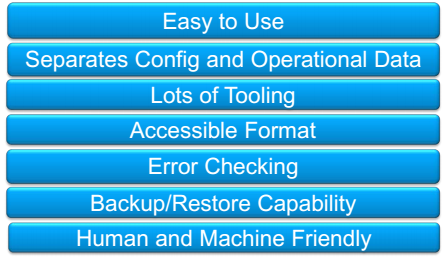
Способ описать, что умеет сетевое устройство и в каком формате ему надо слать запросы







**Преимущества:**



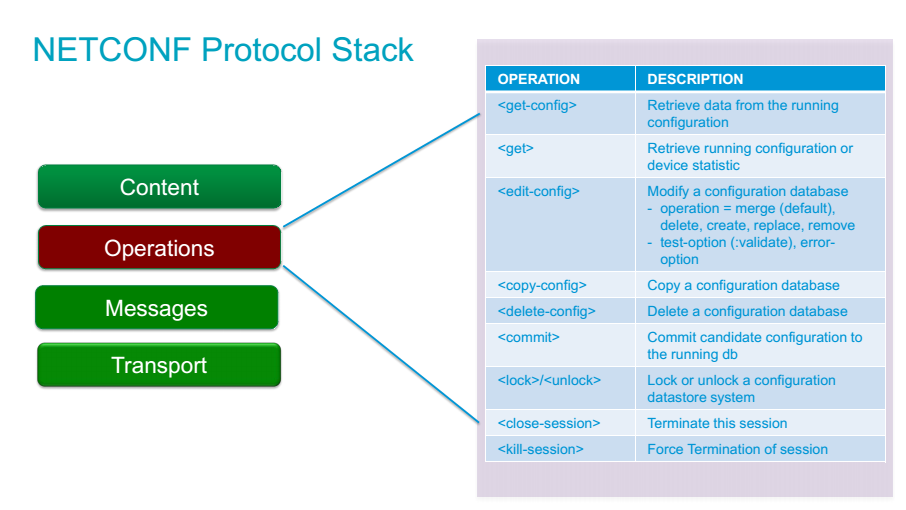
### NETCONF Protocol Stack

**Transport** - SSH

**Messages** - Remote Procedure Call (*RPC*)

**RPC Client’s Request Methods:**

* Controller
* *NMS* (Network management station)
* Script
* Manual



**Content** - Config data, Notification (XML)

**NETCONF Configuration Data Stores**

* Named configuration stores
  + Each data store may hold a full copy of the configuration
* Running is mandatory, Startup and Candidate optional (capabilities :startup, :candidate)
* Running may or may not be directly writable (:writable-running)

**YANG**

* **Data definition language for management data**
  + Original focus on configuration information, but not any more restricted to it
  + Original context: NETCONF
  + Can be separated from NETCONF (not an original goal but important!)
* Consistent data enabled through models defined in a common language
* Model-driven Interfaces

What can you do with YANG

**Define data models for something that you need to manage.** Example:

* Interfaces and their configuration
* Access Control Lists (rules that govern policies how to handle packets)
* Topology

**Specify semantics that allow to validate and maintain consistency**

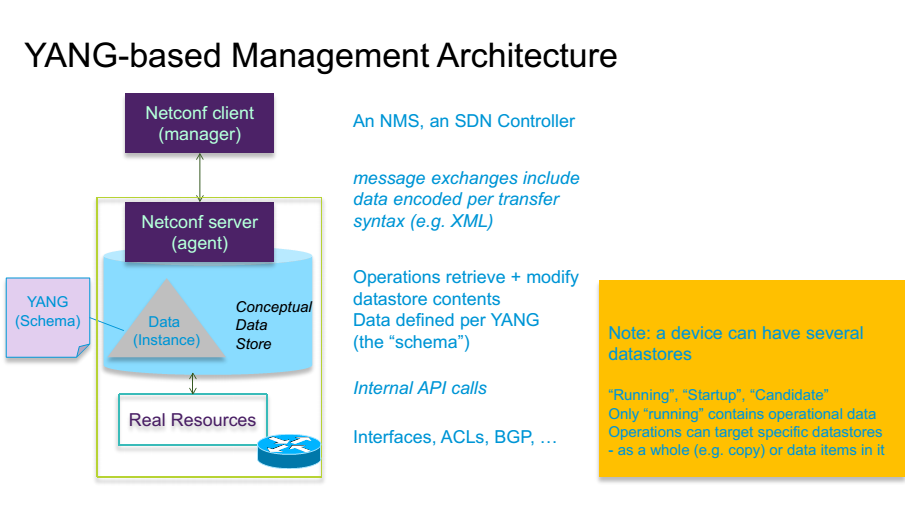
* Operational or configuration – “read-only” or “read-write”
* Conditions and constraints – value dependencies, referential integrity, …
* Units, defaults, ranges, alternatives, …

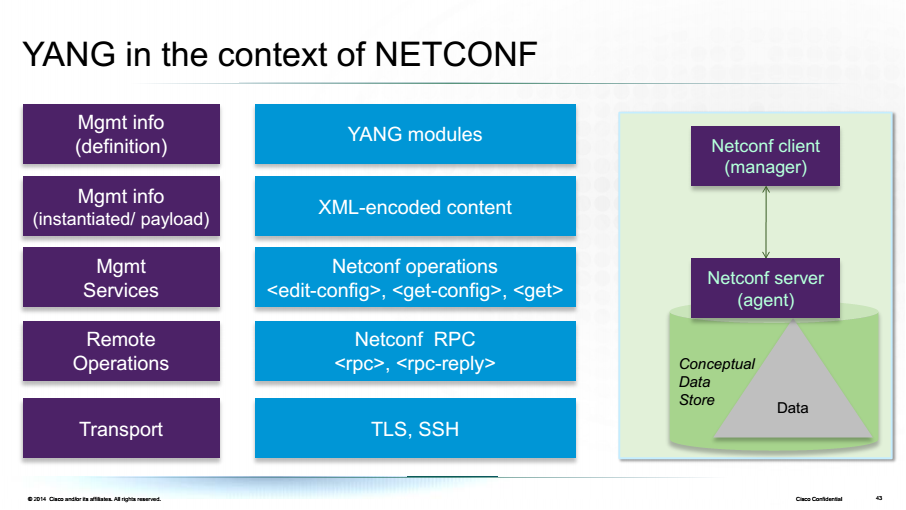
**Extend data models previously defined**

* Simply “insert” new data where applicable (“augmentation”)
* Add a new sub-tree into the existing model

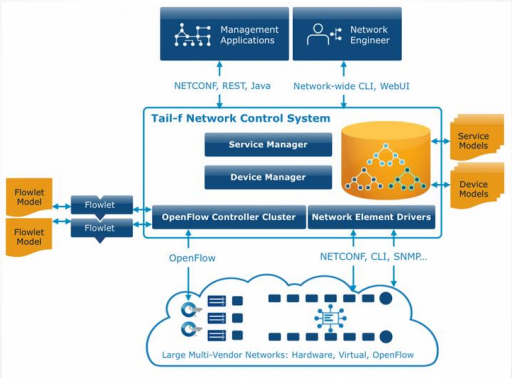
**Instantiate the data as “conceptual datastore” and manage your device by accessing and manipulating contents of that datastore**

* Datastore: an abstraction of a “real resource” – what you are managing





### Tail-F



### NETCONF/YANG хитрый (но логичный шаг) CISCO к управлению своим оборудованием

# 13. Протоколонезависимые SDN коммутаторы: POF.

*POF* - Protocol-Oblivious Forwarding

*FE* - Forwarding Element

### Постановка задачи

**Недостатки OpenFlow:**

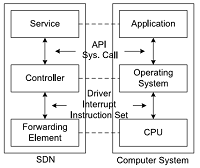
* ориентирован на используемые в сети протоколы (IPv4, IPv6, MPLS) и заголовки (L2, L3, L4) - анахронизм;
* Уровень передачи данных не контролирует состояние потоков (stateless). Не может мониторить статус и изменять поведение потоков без участия контроллера. OF должен иметь возможность управлять более интеллектуальными возможностями аппаратуры.

**Как следствие, это приводит к следующему** :

* По мере появления новых протоколов будут усложняться требования к стандарту и к коммутаторам, что усложнит их дизайн.
* Нет возможности произвольно изменить формат пакета идущего по сети, добавить новые произвольные поля.
* Добавление новых протоколов приведет к необходимости переработки и протокола и ПО коммутатора, а возможно и к переработке аппаратной части, что каждый раз будет приводить к задержкам на время реализации новой опции и тормозить прогресс.
* OF не имеет возможности обрабатывать трафик с учетом L4-L7 сервисов на уровне DataPlane. Даже реализация L2 learning приводит к проблемам производительности и масштабируемости

### Forwarding plane должен быть white box, поведение которого полностью подчиняется контроллеру.

### Аналогия SDN и PC



* FE в SDN должен стать аналогом CPU в PC. (RISC)
* FE не должен знать заранее с каким трафиком ему придется иметь дело.
* Нужно идти по пути упрощения, к более общей модели FE.
* Нужны независимые от протоколов инструкции, которые предлагается назвать Flow Instruction Set (FIS)
* FIS могут быть использованы для разбора пакетов, анализа данных за счет поддержки инструкций, реализующих логику сравнения, ветвления.
* Каждый такой примитив должен быть легко реализуем в платформе.
* Конструирование сетевого сервиса на такой основе аналогично программированию на Ассемблере, что может быть компенсировано в будущем.

**POF FE не обязан понимать формат пакета в отличии от OpenFlow FE.**

**FE должен по указанию контроллера:**

* Анализировать заголовки пакетов на основе указаний контроллера
* Осуществлять поиск по таблицам
* Выполнять соответствующие инструкции

**Взаимодействие с контроллером**

* Контроллер описывает составные части ключа поиска в терминах {offset, length}+{offset, length}…
* Вместо стандартных действий (push vlan, swap mpls…) используются инструкции AddField {offset, length} или задействуются данные из metadata.
* Есть инструкции описывающие математические (Add, Substr, Shift) и логические (AND, OR, NOT, XOR) операции
* Для сложных функций – есть специальные инструкции (IP checksum, TCP checksum), определяя какие данные использовать, какой алгоритм и куда поместить результат.

**Преимущество**

* В случае появления необходимости обработ0ать новый протокол или комбинацию (VXLAN, NVGRE, VLAN+MPLS+PBB) нет необходимости модернизировать стандарт OpenFlow.

### Stateful FE

**Data plane хранит информацию о статусе потока используя flow metadata (не**

**путать с packet metadata)**

* Содержит информацию о потоке (подобно counter и timeout в OF)
* Выделяется не для всех потоков (on-demand)
* Метаданные могут быть использованы сетевыми приложениями.
* Данные могут быть разных типов (Sequence #, time stamp)

**FE может самостоятельно манипулировать потоками (без указания**

**контроллера)**

* По событию или при выполнении заранее определенного условия может быть:
  + создана таблица;
  + создан, удален, обновлен flow в таблице;
* Об этих действиях FE информирует контроллер для синхронизации состояния контроллера и сети.

**Единый набор инструкций подходящий для множества потоков. Вызов инструкций с параметрами.**

* Внутри блока инструкций существует возможность перехода на различные ветви операций с помощью условного и безусловного перехода.

### Дополнения

**Выделение счетчика статистики только по необходимости по запросу контроллера**

* Экономия количества счетчиков и используемой памяти.
* Может быть один счетчик на несколько потоков и несколько счетчиков для одного потока
* Возможность иметь агрегированные счетчики

**Дифференциация типов таблиц (id, capacity, parameters, type):**

* Direct Table (DT)
* Exact Match (EM)
* Longest Prefix Match (LPM)
* Masked match (ММ)

*Возможно дальнейшее расширение в будущем:*

* Range Match (RM)
* RegEx Match (REM)

**Физические и логические таблицы (могут ссылаться на одну физическую)**

* Одни и те же записи могут использоваться несколькими логическими таблицами но с разными инструкциями (IP routing, RPF)
* Несколько логических таблиц могут сосуществовать в рамках одной физической (несколько EM таблиц в одной физической Hash таблице)

### Выводы

**В SDN должна быть более простая и обобщенная модель уровня передачи данных (FE).**

* В рамках модели контроллер должен определять для FE не только ЧТО делать, но и КАК это делать.

**Для сохранения совместимости с текущими версиями необходимо добавить POF инструкции в спецификацию OpenFlow**

* В идеале для поддержки POF должны появиться новые чипы.
* В текущих реализациях *NP* поддержка POF может реализовываться как функция, описывающая POF инструкцию
* В *ASIC* предлагается реализовать POF инструкции в аппаратуре что даст выигрыш в производительности.

**Подчеркивается важность исследовательских работ и работ по стандартизации под эгидой *ONF* в развитии OpenFlow.**

# 14. Протоколонезависимые SDN коммутаторы: P4.

**P4 – прототип дальнейшего совершенствования протокола OpenFlow:**

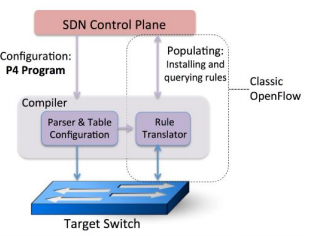
* Контроллер должен уметь менять алгоритм обработки пакетов коммутаторами.
* Коммутаторы не должны быть завязаны на обработку пакетов каких либо сетевых протоколов. Контроллер должен определять:
  + Алгоритм разбора заголовков пакетов (определение типов, имен)
  + Совокупность таблиц (match+action) для обработки этих заголовков
* Программисты должны иметь возможность описывать процесс обработки пакетов независимо от того как реально это будет реализовано в аппаратуре.
* Компилятор должен преобразовывать платформо-независимые описания (на языке P4) в платформо-зависимые воздействия на конкретный коммутатор.

**Нужен гибкий механизм определения используемых при анализе заголовков - новый API (OpenFlow2.0)**

### Архитектура решения

**P4 – высокоуровневый языкпрограммирования пакетных процессоров.**

* P4 – интерфейс взаимодействия контроллера и коммутатора, описывающий как коммутатору обрабатывать пакеты.
* Обработка посредством программируемого парсера (против фиксированного в OF) и множества последовательных или параллельных стадий match+action (в OF только последовательно)



**Операция конфигурирования (Configure) в абстрактной модели:**

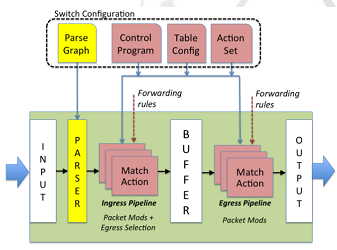
* Программирование парсера
* Определение порядка match+action стадий
* Определение используемых на каждой стадии полей заголовков и производимых с ними действий (проверки и уменьшения для TTL, добавления поля для создания туннеля, проверки контрольной суммы и пр)

**Операция заполнения (Populate) в абстрактной модели:**

* Добавление и удаление записей в match+action таблицы

**Таблицы (match+action) делятся на:**

* Входящие:
  + Определяют исходящий порт + очередь.
  + Пакет может быть отправлен, реплицирован, сброшен, отправлен на контроллер.
* Исходящие:
  + Осуществляют индивидуальную модификацию заголовков (например для multicast)

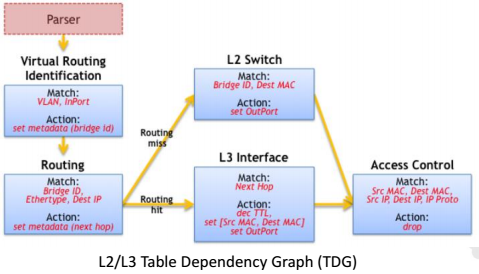


### P4. Описание графа зависимостей между таблицами.

На **1** стадии на (высоком) уровне описывается процесс обработки пакетов на языке P4

На **2** стадии компилятор транслирует P4 в Table Dependency Graph (TDG)

На **3** стадии TDG приводится в соответствие с особенностями коммутатора



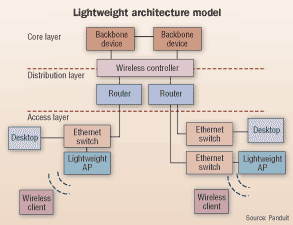
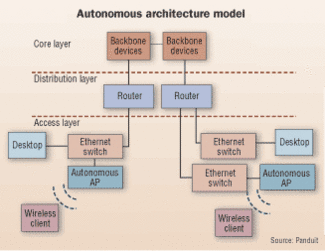
### Выводы

* Протокол должен позволять настраивать коммутаторы на лету “in field”.
* Программист не должен беспокоиться о деталях аппаратной реализации.
* Предложен язык конфигурации.
* Коммутаторы должны стать гибче.

# 15. SDN в беспроводных сетях: CapWap.

CapWap - Control And Provisioning of Wireless Access Points (CAPWAP) (Контроль и администрировании точек беспроводного доступа). Этот протокол позволяет контроллеру *(не пкс-контроллеру)* управлять беспроводными точками доступа.

Есть две архитектуры WLAN (Wireless LAN) - автономная и централизованная.



Автономная - когда все точки доступа работают независимо. Тут сразу куча проблем (со слайда, плюс моя интерпритация, так что может быть ошибка):

* RRM (radio resource management) - это проблемы управления этими точками доступа - вручную настраивать каждую неприятно
* работа с клиентами - отсуствие какой-либо статистики
* “враги” - сложно определять и блочить нежелательные устройства
* проблемы с покрытием - точки доступа могут перекрывать друг друга, и непонятно, к какой точке доступа должен присоединиться пользователь.

Для решения этих проблем решили внедрить контроллер в такую сеть, точки доступа сделать более глупыми и почти все решения принимать на контроллере. *По сути, централизация управления точками доступа - прототип будущего ПКС.*

Цели CapWap:

* обеспечивать централизованное управление различными аппаратными устройствами при развертывании WLAN (wireless LAN)
* сделать конфигурацию различных аппаратных устройств прозрачной, и обеспечить конфигурацию всей сети
* обеспечить мониторинг состояния аппаратных и программных конфигураций для обеспечения правильной работы сети

*(CapWap - это расширение протокола Lightweight Access Point Protocol (LWAPP), который позволяет контроллеру управлять точками доступа. CapWap дополнен установлением безопасного соединения DTLS между контроллером и точками доступа)*

CapWap работает на L3. Необходим IP адрес для точки доступа и адрес контроллера. Соединение происходит по CAPWAP Tunnel, который разработан для работы точек доступа не только WiFi.

*Установление соединения между контроллером и точкой доступа:*

*Request:*

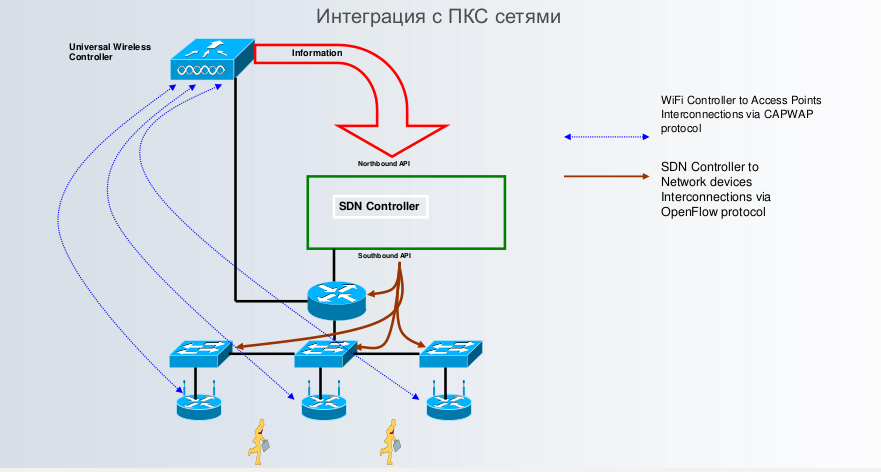
* *тип контроллера и MAC-адрес контроллера*
* *аппаратная и программная версия точки доступа*
* *имя точки доступа, выдавший запрос на вступление*
* *количество и тип радиостанций, присутствующих в точке доступа*
* *полезнуая нагрузка сертификата для создания защищенного подключение*
* *полезная нагрузка сеанса настройки величина сеанса*
* *тестовая полезная нагрузка, чтобы проверить способность сети поддерживать большие кадры*

*Response:*

* *код результата*
* *сертификат ответа полезной нагрузки контроллера*
* *тестовая полезная нагрузка, чтобы проверить способность сети поддерживать большие кадры*

*Что-то очень мутное, но один слайд об этом есть:*

*У протокола два режима работы: Split-MAC и Local-MAC. Local-MAC - все беспроводные функции MAC (L2) выполняются на точке доступа, включая управление и обработка кадров управления - речь о функциях “реального времени”. Split-MAC разделяет обработку функций между контроллером и точкой доступа. В режиме реального времени MAC функции включают в себя функции, такие как передача сигнала и получение ответа, обработки кадра управления (например, отправки запроса на передачу), повторную передачу, и так далее. Функции не в реальном времени включают аутентификацию и деаутентификации; ассоциации и реассоциация.*

Описанную выше систему (WLAN контроллер + точки доступа) можно интегрировать с ПКС-сетью. 

Преимущества интегрированных решений СapWap + SDN:

- Никаких избыточных туннелей *- не понял, что это значит*

- Нет неоптимальных путей - *не понял, что это значит*

- Возможность реализовать удобные геосервисы (контроллер WiFi информирует о местонахождении пользователя и приложение настраивает все сетевые устройства для трафика этого пользователя)



*WTP (Wireless transaction protocol, операционный протокол беспроводного доступа) — стандарт, используемый в беспроводной связи (в частности, в Wi-Fi и в мобильной телефонии)--*

# 16. SDN в беспроводных сетях: SDR.

SDR - Software-defined radio.

Программно-определяемая радиосистема — радиопередатчик и/или радиоприемник, использующий технологию, позволяющую с помощью программного обеспечения устанавливать или изменять рабочие радиочастотные параметры, включая, в частности, диапазон частот, тип модуляции или выходную мощность, выполнять ЦАП и АЦП. Заказчики: Военные, криминал.

Т.е. раньше для каждой задачи нужно было создавать свою строго настроенную железку с “зашитыми” параметрами (частота, модуляция и т.д.), а теперь можно принимать произвольные сигналы и обрабатывать их программно.

В простейшем случае приёмник для SDR антенна непосредственно подключается к АЦП. Цифровой сигнальный процессор считывает сигнал с преобразователя и программно преобразует его в любую форму.

*В лекции есть несколько слайдов про амплитудную, частотную и фазовую модуляцию, расписывать это не буду, но, может, имеет смысл повторить.*

Практическая польза:

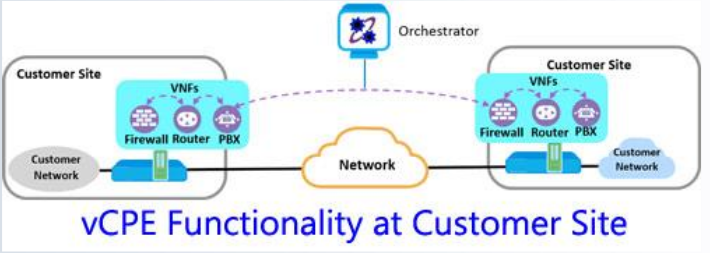
* в промышленности построение радиосистем упростилось в сотни раз
* почти любая вычислительная система готова решать задачи обработки радиосигнала, нужно только завести его внутрь!

Пример сервиса: <http://www.radioscanner.ru/>

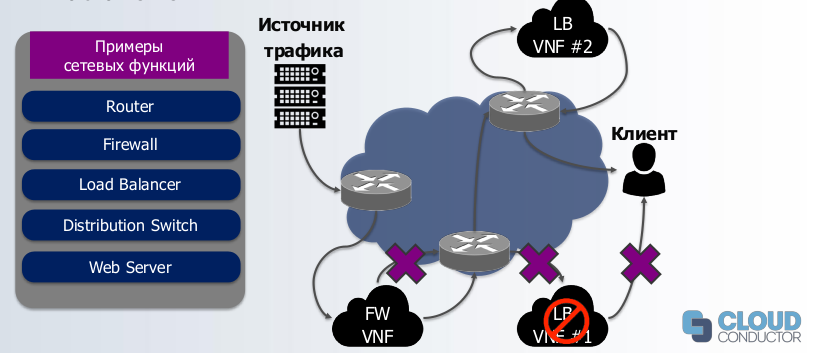
# 17. Примеры практического применения SDN&NFV в России.

*Я бы рекомендовал глянуть в слайды лекции, потому что все что тут написано хромает на полторы ноги. В слайдах - просто картинки, по ним сложно понять что-то и непонятно, как это относится к России. Поэтому, я думаю, ответ на этот вопрос - это импровизация на тему применения SDN и NFV в принципе с оговоркой на то, что в России все збс с этим добром.*

NFV, например, активно применяется телеком-операторами, например, используя vCPE - Virtual Customer Premises Equipment (Виртуальное Абонентское оборудование), тем самым сокращая количество и стоимость физических устройств, которые обеспечивали эти функции (FireWall, Routing). Конечные абоненты могут использовать универсальные устройства «вечные», управляемые оркестратором провайдера:



Сloud Conducter - Self Organizing Cloud облачная платформа от ЦПИКС. В ней активно применяется виртуализация сетевых функций: баланлировка нагрузки, фаерволы и т.д :



Понятно, что в России SDN - это RunOS, сверхконтроллер, супер-производительный! 30 млн событий в секунду! + офигенный набор приложений для управлению сетью!. С помощью вебки в режиме реального времени можно мониторить трафик и конфигурировать сеть (менять ширину полосы пропускания, задержки) - вот такие вот они сказочные ребята. Это может быть использовано как в корпоративном сегменте, так и в домашней сети (у рунос есть REST-api)

Не забываем про интеграцию SDN-контроллера с WIFI-контроллером. Нам видос показывал в прошлом семе Шалимов, значит это есть, и это работает. С помощью этого, можно предоставлять различные геосервисы (*Если обязать пользователей корпоративной сети ВСЕГДА держать интерфейс Wi-Fi, даже работая через провод, мы всегда будем знать его местоположение в офисе (3-5 м.) Таким образом, мы сможем динамически изменять правила, основанные на географическом положении! Например: финансовые документы будут приниматься с ноутбуком, только если он находится в бухгалтерской комнате*)

И, наверное, нужно что-то сказать про NFV + SDN:



