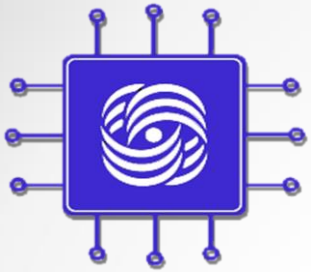


SDN&NFV: SDN/OpenFlow контроллеры и приложения

Доп. главы Компьютерных сетей и телекоммуникации

к.ф.-м.н., м.н.с., Шалимов А.В.

ashalimov@lvk.cs.msu.su

@alex_shali

@arccnnews

Часть 0: SDN/OpenFlow

Что такое SDN/OpenFlow?

SDN = **S**oftware **D**efined **N**etworking

Внедрения

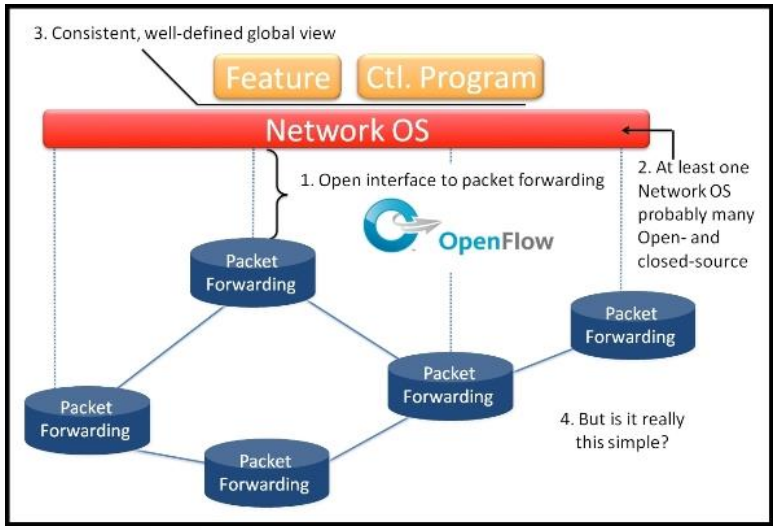
Google

NTT Communications

-
-
-

Основные принципы

- Физическое разделение уровня передачи данных от уровня управления сетевых устройств.
- Логически централизованное управление.
- Программируемость.
- Открытый единый интерфейс управления.

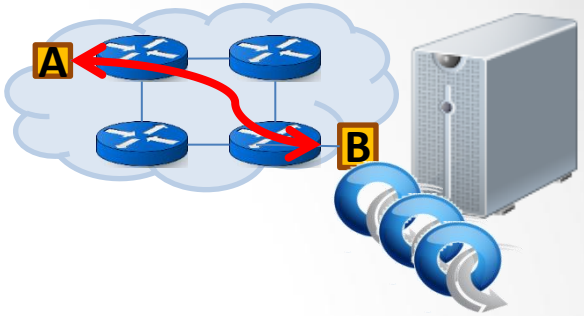


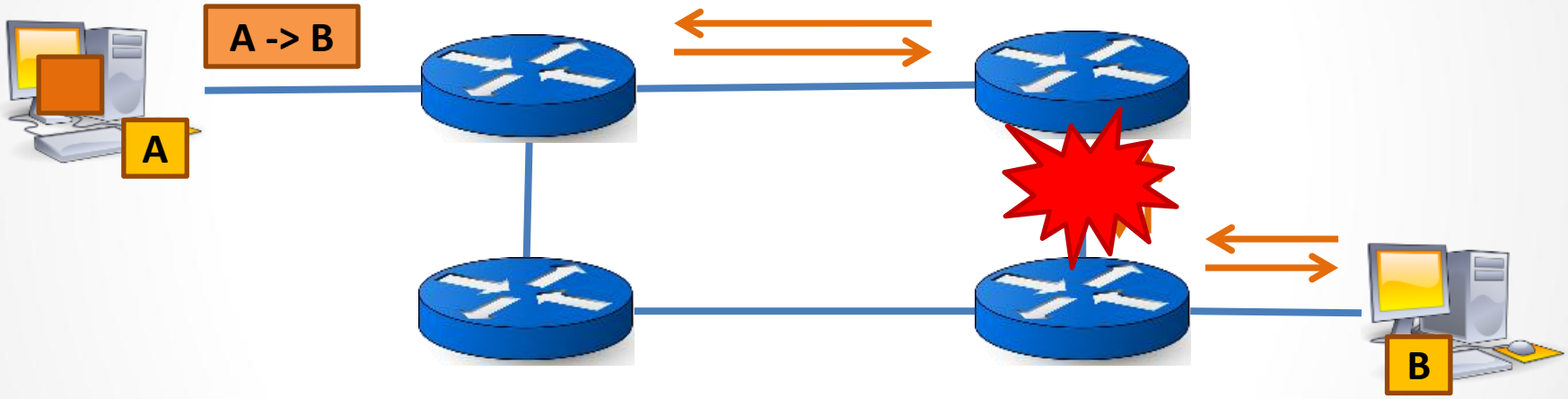
Преимущества

- Упрощение управления сетью (OPEX)
- Удешевление оборудования (CAPEX)
- Разработка ранее недоступных сервисов

"SDN means thinking differently about networking"

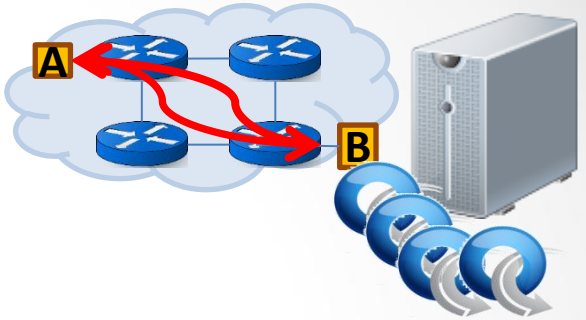
Маршрутизация с SDN/OpenFlow

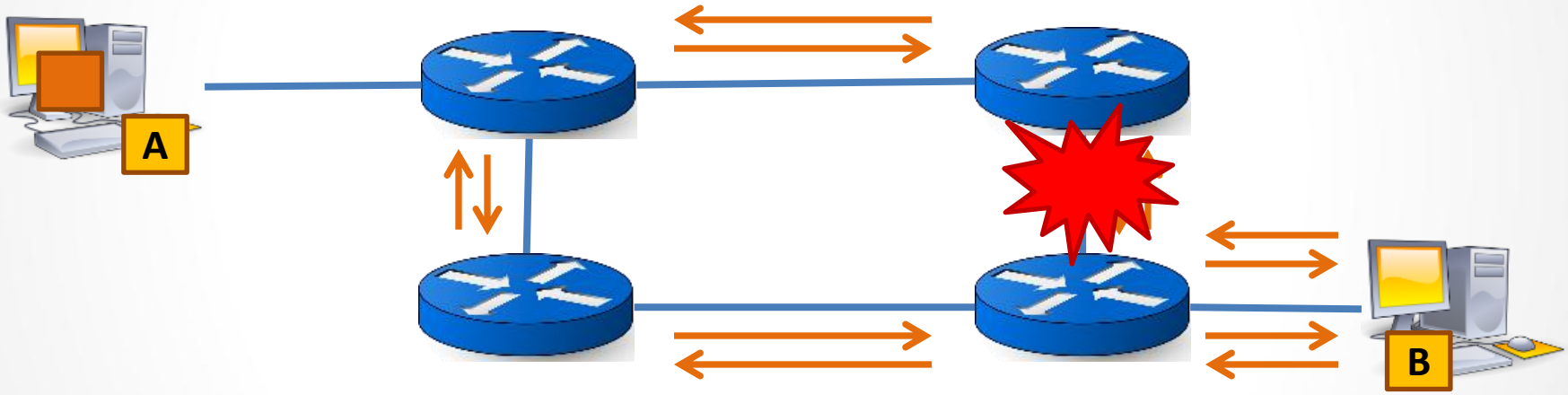




- Неизвестный пакет отправляется на контроллер (OF_PACKET_IN).
- Контроллер вычисляет лучший маршрут через всю сеть (с наименьшей стоимостью и удовлетворяющий политикам маршрутизации).
- Соответствующие правила OpenFlow устанавливаются на коммутаторы + сразу и обратный маршрут (OF_PACKET_OUT/FLOW_MOD).

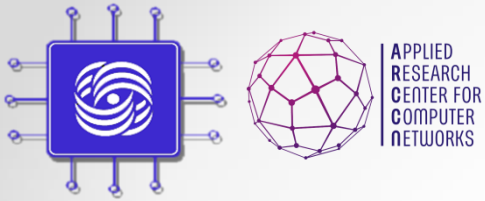
Маршрутизация с SDN/OpenFlow





- Неизвестный пакет отправляется на контроллер (OF_PACKET_IN).
- Контроллер вычисляет лучший маршрут через всю сеть (с наименьшей стоимостью и удовлетворяющий политикам маршрутизации).
- Соответствующие правила OpenFlow устанавливаются на коммутаторы + сразу и обратный маршрут (OF_PACKET_OUT/FLOW_MOD).
- **Динамическая переконфигурация в случае ошибки сети.**

Часть I: OpenFlow контроллеры



OpenFlow контроллер

- 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**.

Список OpenFlow контроллеров

- Их действительно много
  - Nox, Pox, MUL, Ruy, Beacon, OpenDaylight, Floodlight, Maestro, McNettle, Flower, Runos
  - Разные языки программирования от Python до Haskell, Erlang
- Для образования - Pox.
- Два больших комьюнити
  - ONOS (Stanford)
  - OpenDayLight (Cisco)
- В России – наш Runos
  - arccn.github.io/runos

Архитектура OpenFlow контроллера

Требования к контроллеру ПКС

- Производительность
  - Пропускная способность
    - events per second
  - Задержка
    - us
- Надежность и безопасность
  - 24/7
- Программируемость
  - Функциональность: приложения и сервисы
  - Интерфейс программирования

- ЦОД требует обработку >10М событий в секунду
- Реактивные контроллеры более “чувствительные”

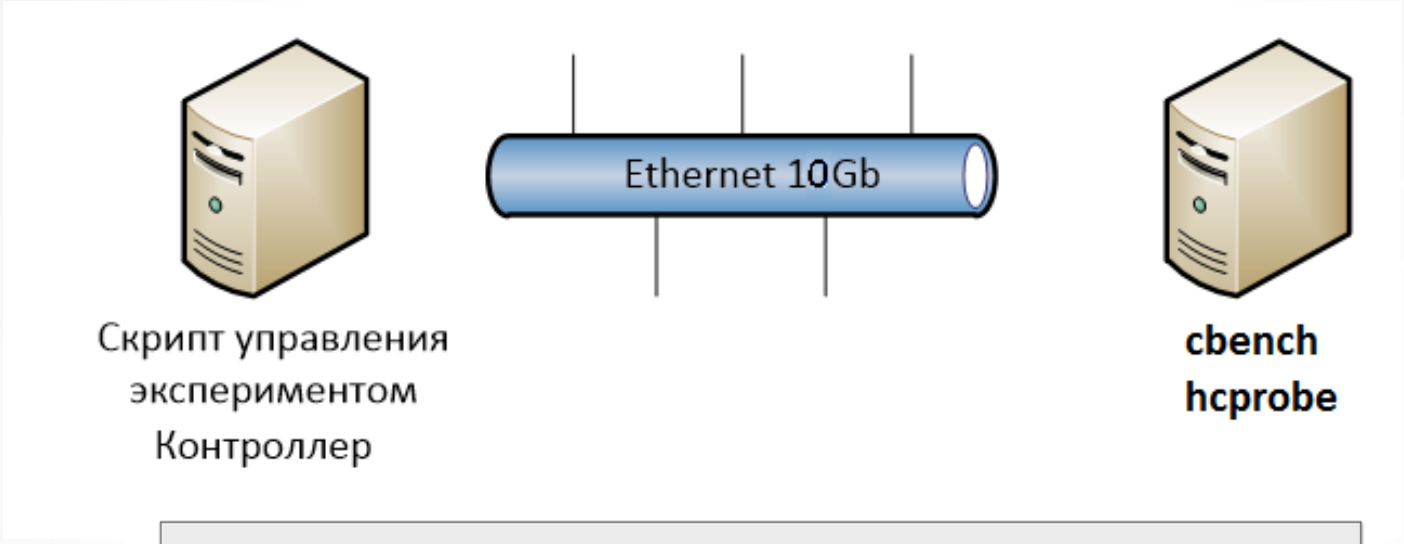
Полный список контроллеров (2013)

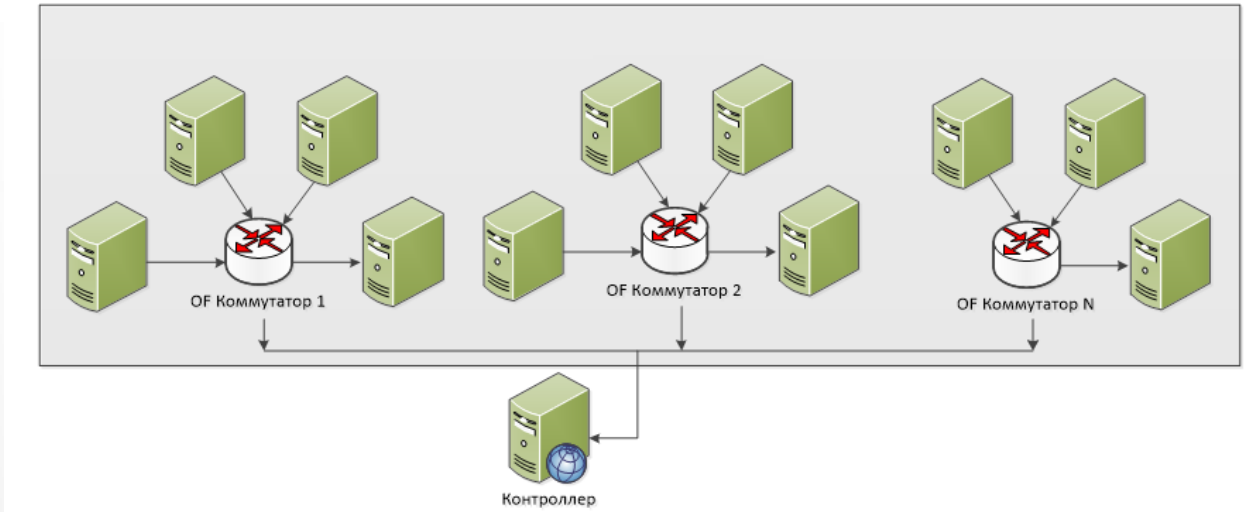
- NOX-Classic
- NOX
- Beacon
- Floodlight
- SNAC
- Ryu
- POX
- Maestro
- Trema
- Helios
- FlowER
- MUL
- McNettle
- NodeFlow
- Onix
- SOX
- Kandoo
- Jaxon
- Cisco ONE controller
- Nicira NVP Controller
- Big Network Controller
- IBM Programmable Network Controller
- HP SDN Controller
- NEC Programmable Flow

Экспериментальное исследование

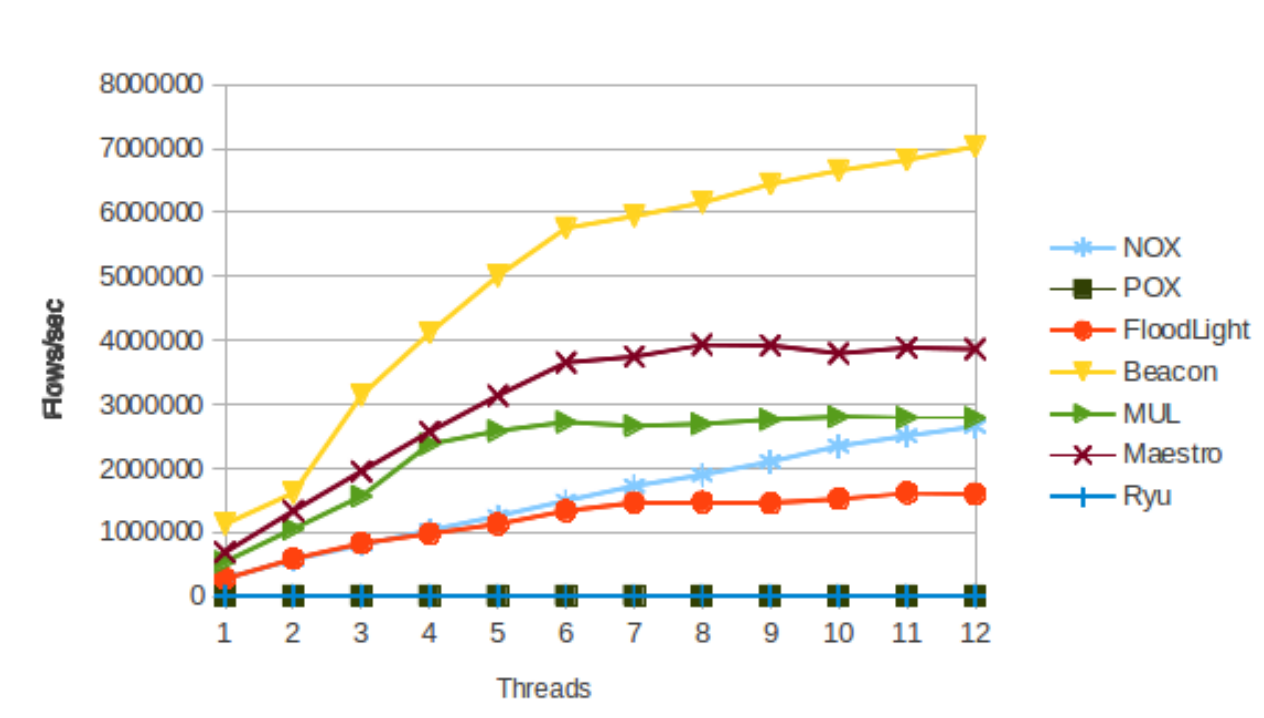
- Производительность
  - максимальное количество запросов на обработку
  - время обработки запроса при заданной нагрузке
- Масштабируемость
  - изменение показателей производительности при увеличении числа соединений с коммутаторами и при увеличении числа ядер процессора
- Надежность
  - количество отказов за время тестирования при заданном профиле нагрузок
- Безопасность
  - устойчивость к некорректно сформированным сообщениям протокола OpenFlow

Стенд


Ethernet 10Gb
Скрипт управления экспериментом
Контроллер
cbench
hcprobe


OF Коммутатор 1
OF Коммутатор 2
OF Коммутатор N
Контроллер

Результаты сравнения (2013)



- Максимальная производительность **7 000 000 потоков в секунду**.
- Минимальное время задержки **от 50 до 75 мкс**.
- Недостатки:
  - Надежность контроллеров вызывала вопросы
  - Производительность была не достаточна (DC >10M fps)

Часть II: Производительность и программируемость OpenFlow контроллеров

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.

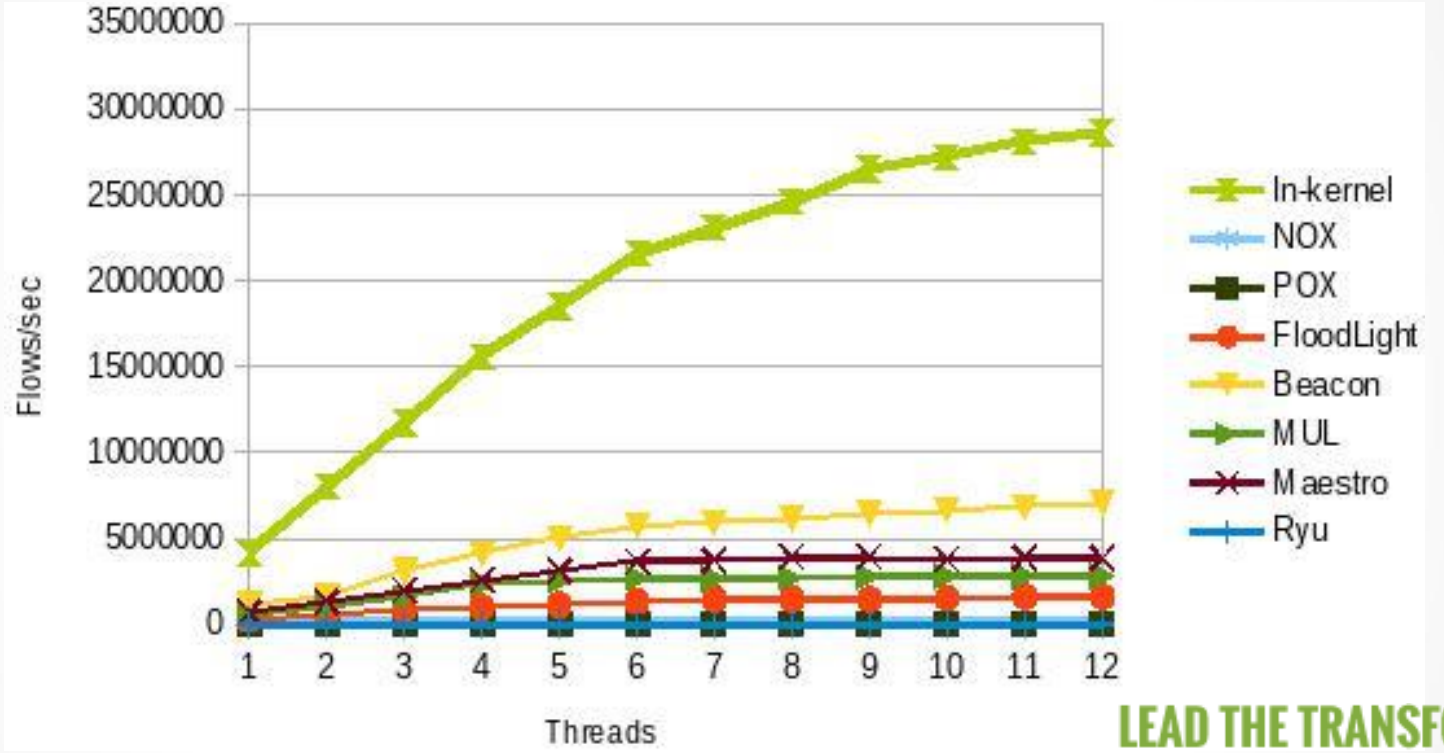
Пример: In-kernel контроллер

Контроллер был реализован в ядре ОС Linux [2]

- 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

[2] P. Ivashchenko, A. Shalimov, R. Smeliansky "*High performance in-kernel SDN/OpenFlow controller*", Proceedings of the 2014 Open Networking Summit Research Track, USENIX, March 3-5, 2014 Santa Clara, USA

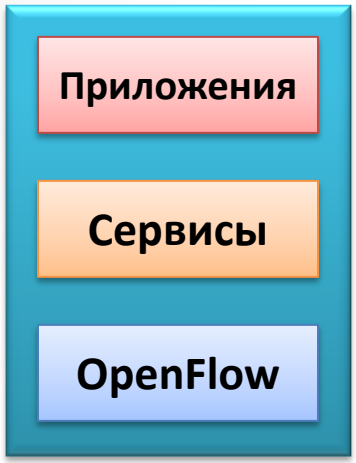
Производительность (kernel)



- Производительность равна **30M fps**
- Задержка **45us**

RUNOS – это линейка OpenFlow контроллеров

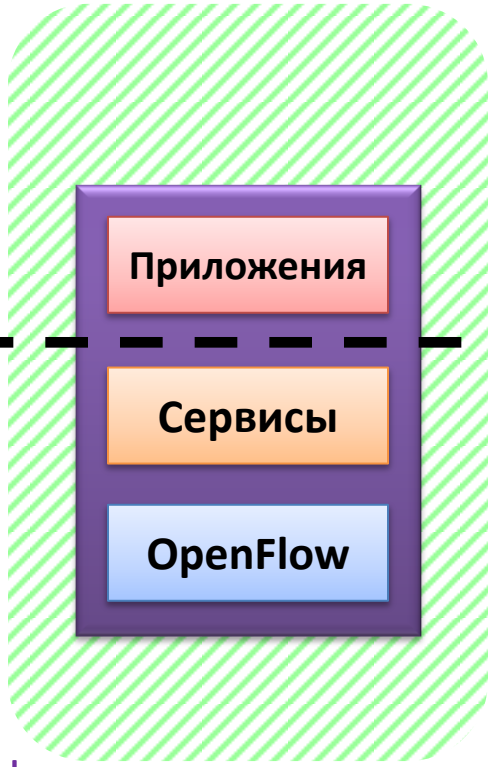
+: большая фунци-ть
-: низкая произв-ть



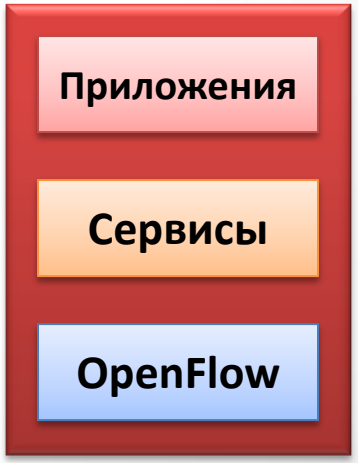
NOX, Pox, Floodlight, OpenDaylight, MUL, etc

User space
Kernel space

Что может быть по середине?



+: большая фунци-ть
+: высокая произв-ть



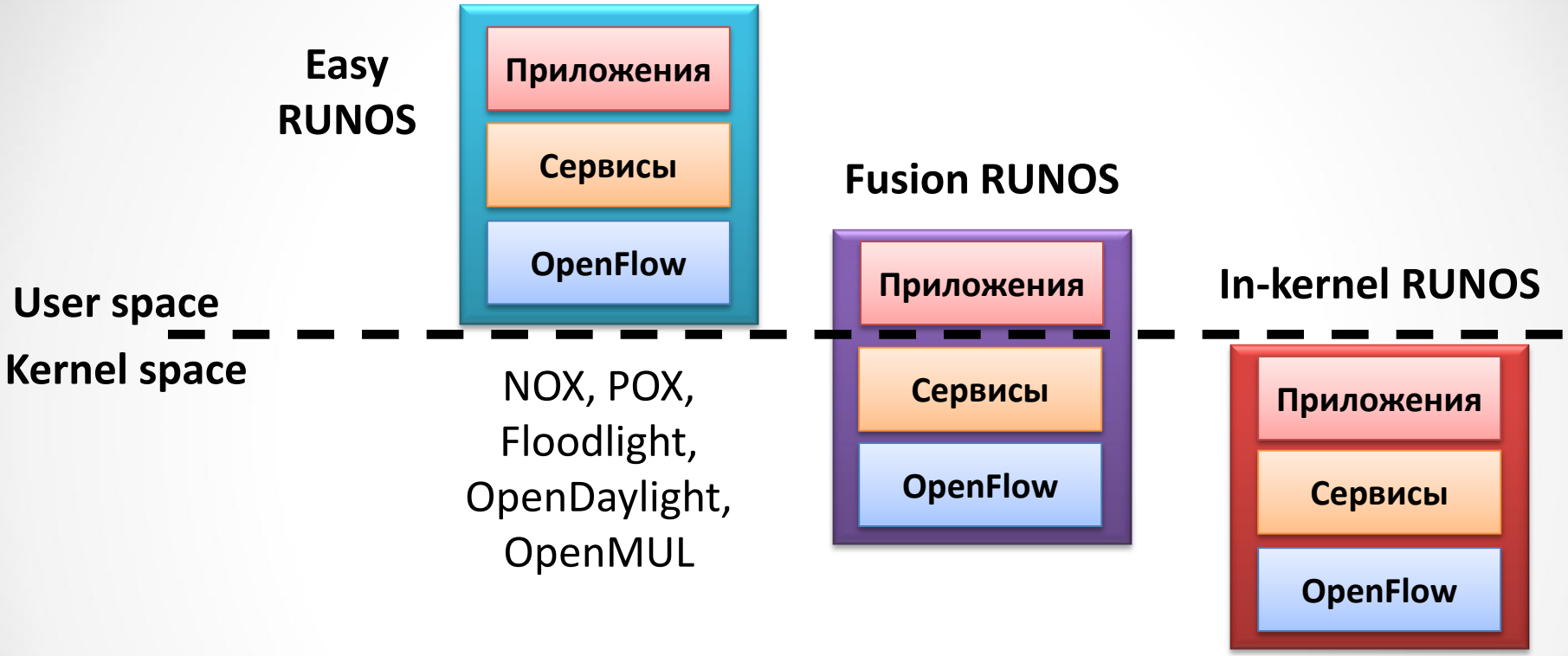
In-kernel

+: высокая произв-ть
-: мало функций

Архитектура:

- Прилож-я в userspace
- Часто используемые сервисы в ядре (топология)
- Интерфейс взаимодействия

Варианты исполнения RUNOS

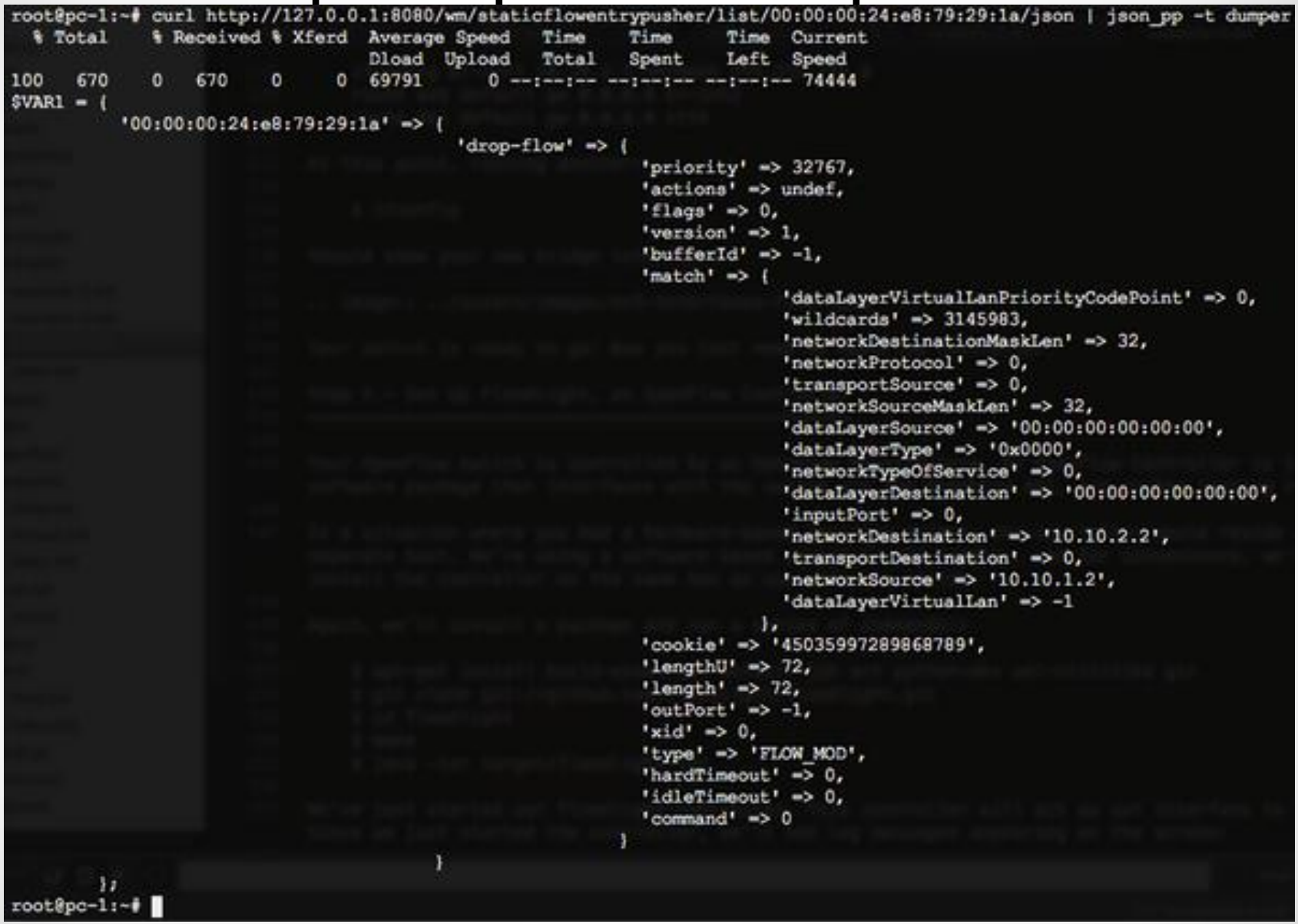


	Easy	Fusion	In-kernel
Функциональность	+	+	-
Производительность	Низкая	Высокая	Высокая

Программируемость

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

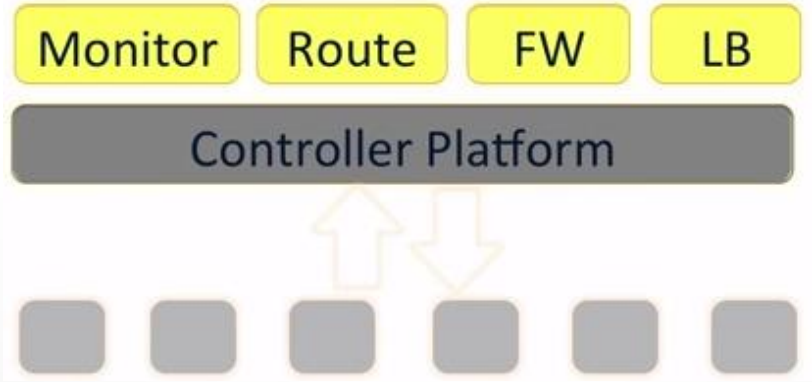
Пример REST запроса



```
root@pc-1:~# curl http://127.0.0.1:8080/wm/staticflowentrypusher/list/00:00:00:24:e8:79:29:1a/json | json_pp -t dumper
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   670    0   670    0     0  69791      0 --:--:-- --:--:-- --:--:-- 74444
$VAR1 = {
          '00:00:00:24:e8:79:29:1a' => {
                                         'drop-flow' => {
                                                          'priority' => 32767,
                                                          'actions' => undef,
                                                          'flags' => 0,
                                                          'version' => 1,
                                                          'bufferId' => -1,
                                                          'match' => {
                                                                       'dataLayerVirtualLanPriorityCodePoint' => 0,
                                                                       'wildcards' => 3145983,
                                                                       'networkDestinationMaskLen' => 32,
                                                                       'networkProtocol' => 0,
                                                                       'transportSource' => 0,
                                                                       'networkSourceMaskLen' => 32,
                                                                       'dataLayerSource' => '00:00:00:00:00:00',
                                                                       'dataLayerType' => '0x0000',
                                                                       'networkTypeOfService' => 0,
                                                                       'dataLayerDestination' => '00:00:00:00:00:00',
                                                                       'inputPort' => 0,
                                                                       'networkDestination' => '10.10.2.2',
                                                                       'transportDestination' => 0,
                                                                       'networkSource' => '10.10.1.2',
                                                                       'dataLayerVirtualLan' => -1
                                                                     },
                                                          'cookie' => '450359972898688789',
                                                          'lengthU' => 72,
                                                          'length' => 72,
                                                          'outPort' => -1,
                                                          'xid' => 0,
                                                          'type' => 'FLOW_MOD',
                                                          'hardTimeout' => 0,
                                                          'idleTimeout' => 0,
                                                          'command' => 0
                                                        }
                                       }
        };
root@pc-1:~#
```

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



Низкоровневые детали OpenFlow

A.3.4.1 Modify Flow Entry Message

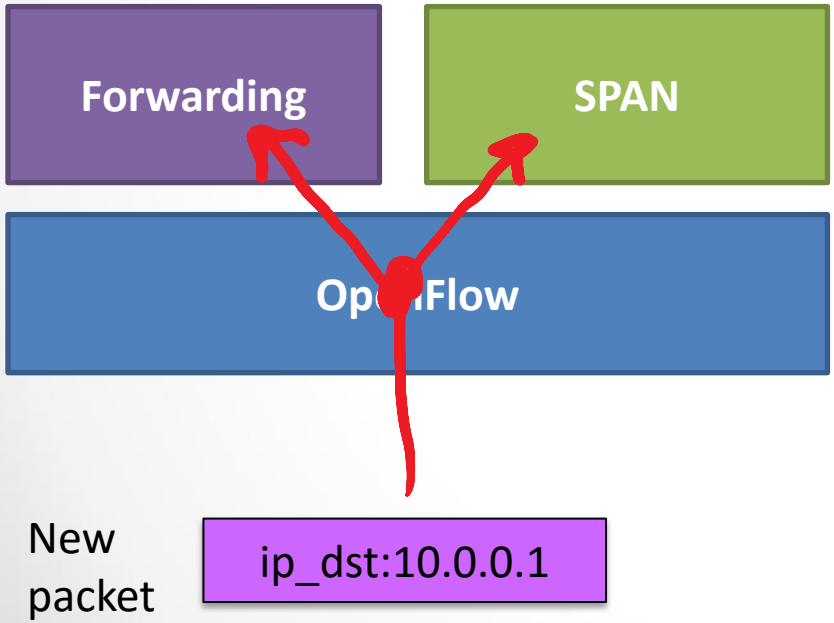
Modifications to a flow table from the controller are done with the OFPT_FLOW_MOD message:

```
/* Flow setup and teardown (controller -> datapath). */
struct ofp_flow_mod {
    struct ofp_header header;
    uint64_t cookie;              /* Opaque controller-issued identifier. */
    uint64_t cookie_mask;         /* Mask used to restrict the cookie bits
                                     that must match when the command is
                                     OFPFC_MODIFY* or OFPFC_DELETE*. A value
                                     of 0 indicates no restriction. */

    /* Flow actions. */
    uint8_t table_id;              /* ID of the table to put the flow in.
                                      For OFPFC_DELETE_* commands, OFPTT_ALL
                                      can also be used to delete matching
                                      flows from all tables. */
    uint8_t command;               /* One of OFPFC_*. */
    uint16_t idle_timeout;         /* Idle time before discarding (seconds). */
    uint16_t hard_timeout;         /* Max time before discarding (seconds). */
    uint16_t priority;             /* Priority level of flow entry. */
    uint32_t buffer_id;            /* Buffered packet to apply to, or
                                      OFP_NO_BUFFER.
                                      Not meaningful for OFPFC_DELETE*. */
    uint32_t out_port;             /* For OFPFC_DELETE* commands, require
                                      matching entries to include this as an
                                      output port.  A value of OFPP_ANY
                                      indicates no restriction. */
    uint32_t out_group;            /* For OFPFC_DELETE* commands, require
                                      matching entries to include this as an
                                      output group.  A value of OFPG_ANY
                                      indicates no restriction. */
    uint16_t flags;                /* One of OFPFF_*. */
    uint8_t pad[2];
    struct ofp_match match;        /* Fields to match. Variable size. */
    //struct ofp_instruction instructions[0]; /* Instruction set */
};
OFP_ASSERT(sizeof(struct ofp_flow_mod) == 56);
```

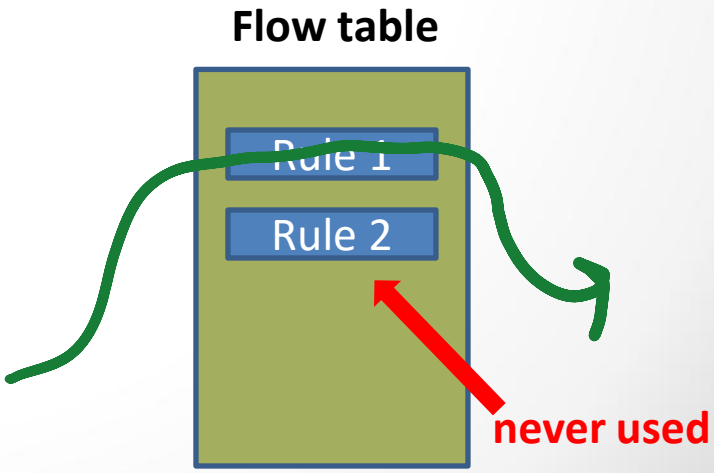
Possible problems in OpenFlow controllers

Example of **the problem** with running several apps independently:

- Forwarding and Span apps. First app sends a flow over port 1, while second ones sends the same flow over port 5. Rules intersect with each other.
- Final rules order in the flow table is unknown.
- Packets will go using only the first rule. Thus, only one app will work. **Conflict!**
- We may to resolve such conflicts and some others. **Just ip_src:10.0.0.1 -> output:1,5!**

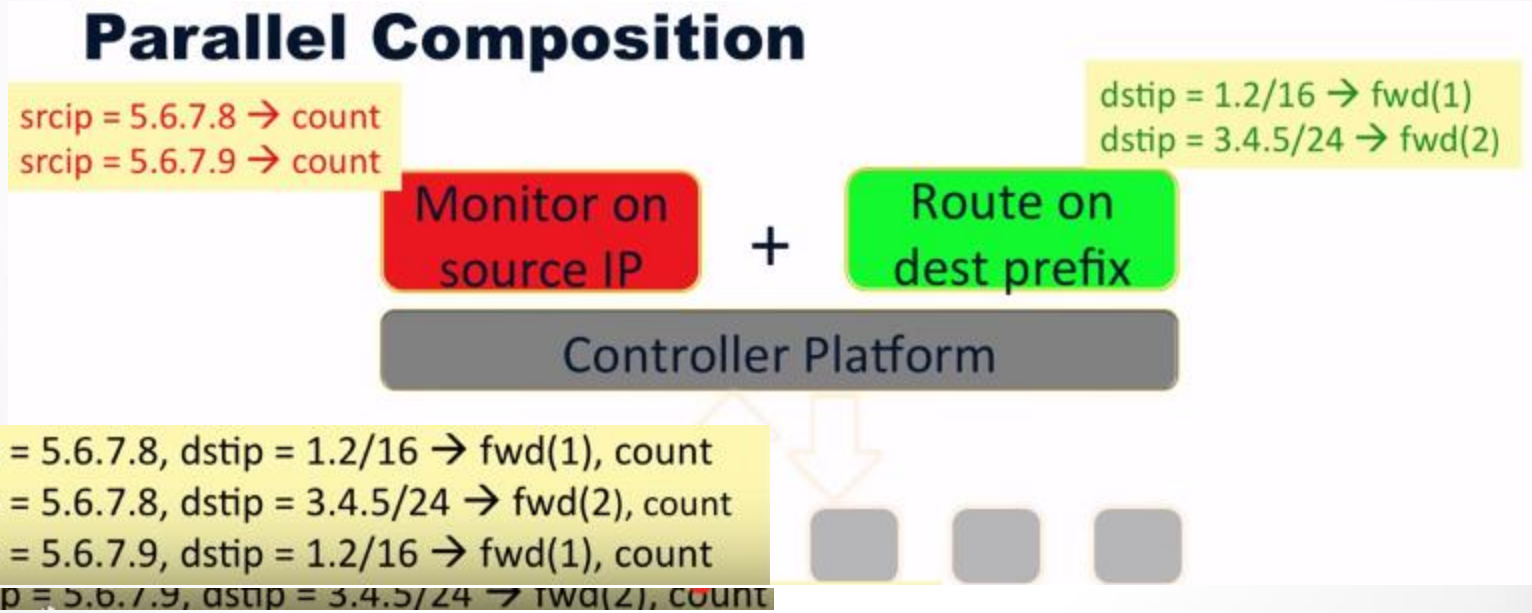


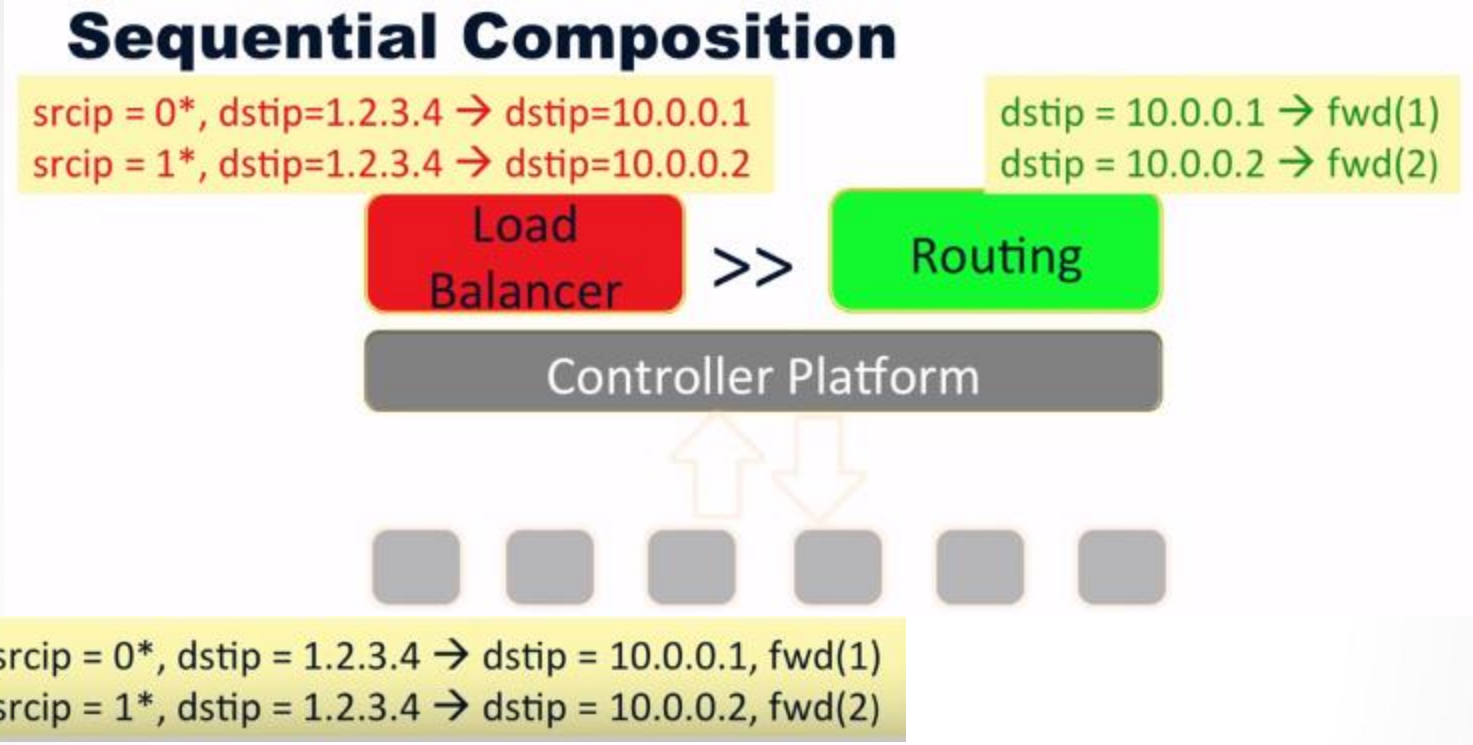
Rule 1: ip_src:10.0.0.1 -> output:1
Rule 2: ip_src:10.0.0.1 -> output:5



Композиция 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)





- **Pyretic**
  - (match(dstip=2.2.2.8) >> fwd(1)) +
    (match(dstip=2.2.2.9) >> fwd(2))

Pyretic пример LB

As another example, consider a server load-balancing policy that splits request traffic directed to destination 1.2.3.4 over two backend servers (2.2.2.8 and 2.2.2.9), depending on the first bit of the source IP address (packets with sources starting with 0 fall under IP prefix 0.0.0.0/1 and are routed to 2.2.2.8). This results in the policy:

```
L = match(dstip='1.2.3.4') >>
    ((match(srcip='0.0.0.0/1') >> modify(dstip='2.2.2.8')) +
     (~match(srcip='0.0.0.0/1') >> modify(dstip='2.2.2.9')))
```

- http://frenetic-lang.org/publications/pyretic-login13.pdf

Часть III: Runos контроллер

Сетевая операционная система RUNOS

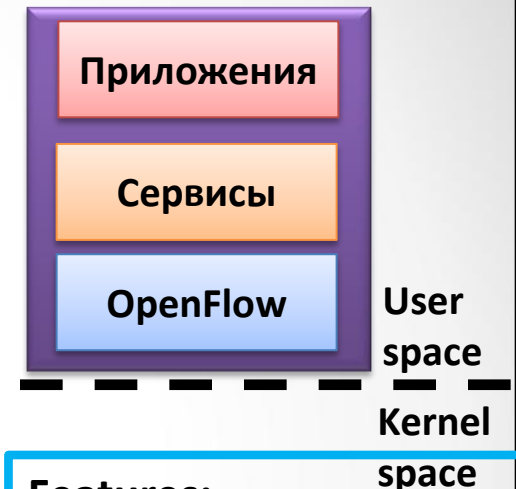
RUNOS = **RU**ssian **N**etwork **O**perating **S**ystem

- Цель проекта
  - *“Could an OpenFlow controller be both easy to develop applications for and also high performance?”*
  - Разработать систему, которая будет удобна для разработки новых сетевых приложений
  - При этом помнить о быстроте

Коммутатор	Версия OpenFlow
Arista 7050T-52	OF1.0
NEC PF5240F	OF1.0, OF1.3
Extreme X460-24p	OF1.0, OF1.3
OpenvSwitch 1.6 и выше	OF1.0, OF1.3
Brocade	OF 1.3
Huawei	OF 1.3

RUNOS: особенности

- Проблема запуска нескольких приложений, интеграция с приложениями других разработчиков
  - требуется статическая подстройка приложений под себя, порядок и способ передачи информации между ними.
  - нет механизма контроля и разрешения конфликтов между приложениями (генерация пересекающихся правил).
- В RUNOS стоит задача решить указанные выше проблемы:
  - часть настройки происходит автоматически по мета информации, связывание происходит динамически
  - разработана система разрешения конфликтов
  - Широкий набор сервисов для упрощения разработки новых приложений



Features:

- Algorithmic policies (rule generation)
- Client-friendly API using EDSL grammar (low level details are hidden inside the runtime – overloading, templates)
- Modules composition (parallel and sequential composition)

Параметры запуска

Config (json):

```
“controller”: {
  “threads”: 4
},
“loader”: {
  “threads”: 3
},
“link discovery”: {
   “poll-interval” : 10,
   “pin-to-thread” : 2
},
“learning switch”: {
}
…
```

- Задается количество нитей контроллера
  - для взаимодействия со свитчами
  - для работы приложений
- Список приложений
  - их параметры (poll-interval)
  - зафиксировать нить выполнения или выделить в монопольное пользование (pin-to-thread, own-thread)

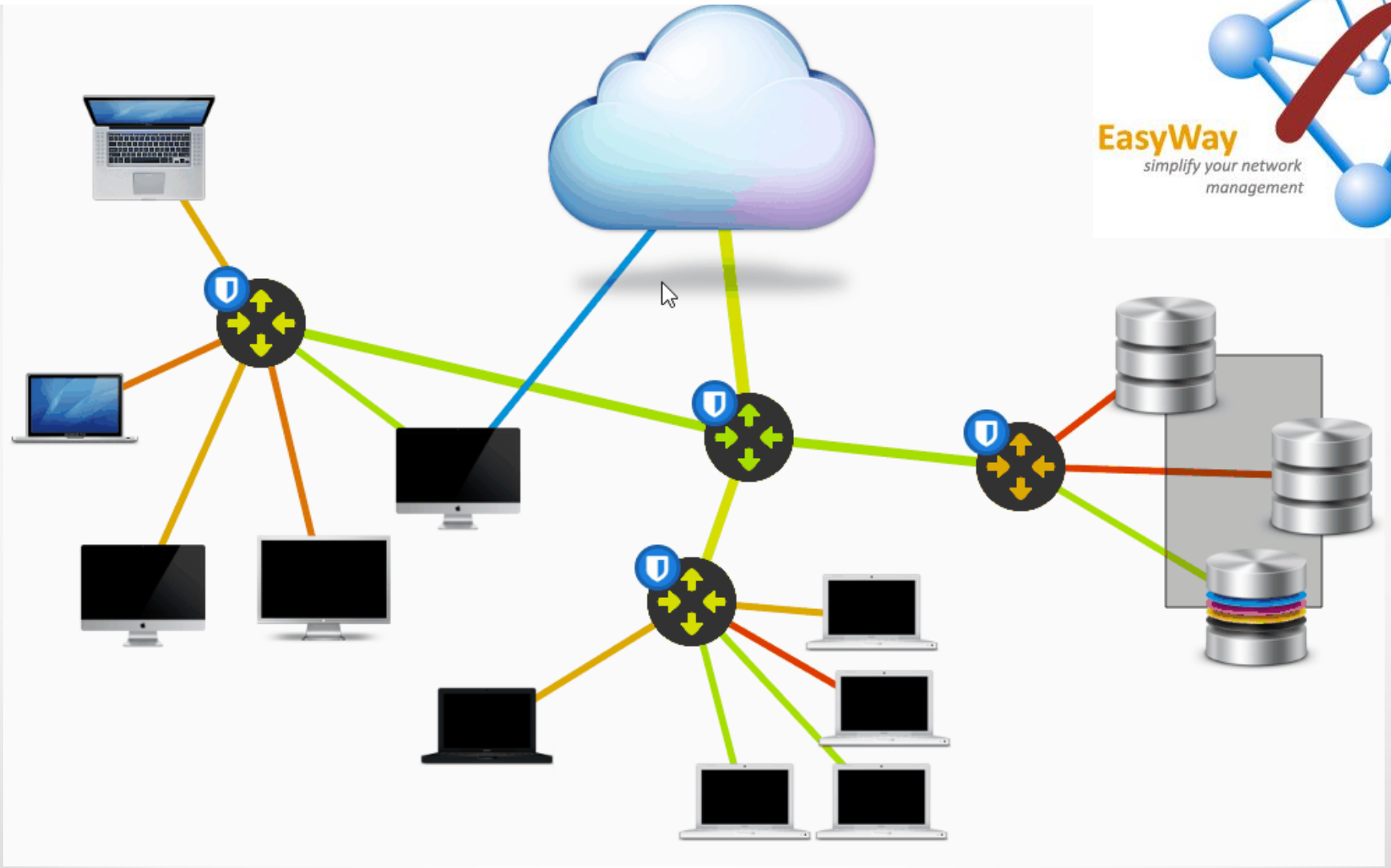
Реализация

Ключевые слова: C++11, QT, Boost (asio, proto, graph

Основные сторонние компоненты:

- **libfluid project** (_base, _msg)
  - для взаимодействия со свитчами и разбор OpenFlow 1.3 сообщений
- **libtins**
  - разбор пакетов внутри OpenFlow сообщений
- **glog** (google log)
  - логирование, многопоточное
- **tcmalloc** (google performance tools)
  - альтернативная более быстрая реализация malloc/free
- **json11**
  - разбор конфигурационного файла
- **boost graph**
  - Хранение топологии, поиск маршрута

Пример графического интерфейса EasyWay



Описания релизов

Сейчас версия **0.5**

- ядро контроллера
- построение топологии
- построение маршрута через всю сеть
- первая версия системы генерации правил
- Rest API (совместимый с Floodlight)
- WebUI (мониторинг загрузки, просмотр таблиц, удаление и добавление правил)
- Проактивная загрузка правил
- Холодное резервирование
- ARP кеширование

Описания релизов

Версия **0.6** - следующий большой релиз (конец сентября)

- *Полное обновление структуры ядра контроллера*. Нет привязка к конкретной версии протокола OpenFlow. Своя модель, расширяемая под любые новые поля, в том числе и специфические для оборудования.
- *Пакетная грамматика для сетевых приложений*. Упрощает разработку новых приложений.
- *Обновление системы генерации правил* — повышена скорость работы и улучшена генерация правил (по количетсву правил и числу приоритетов).
- Возможность статического связывания модулей.
- Система тестов.

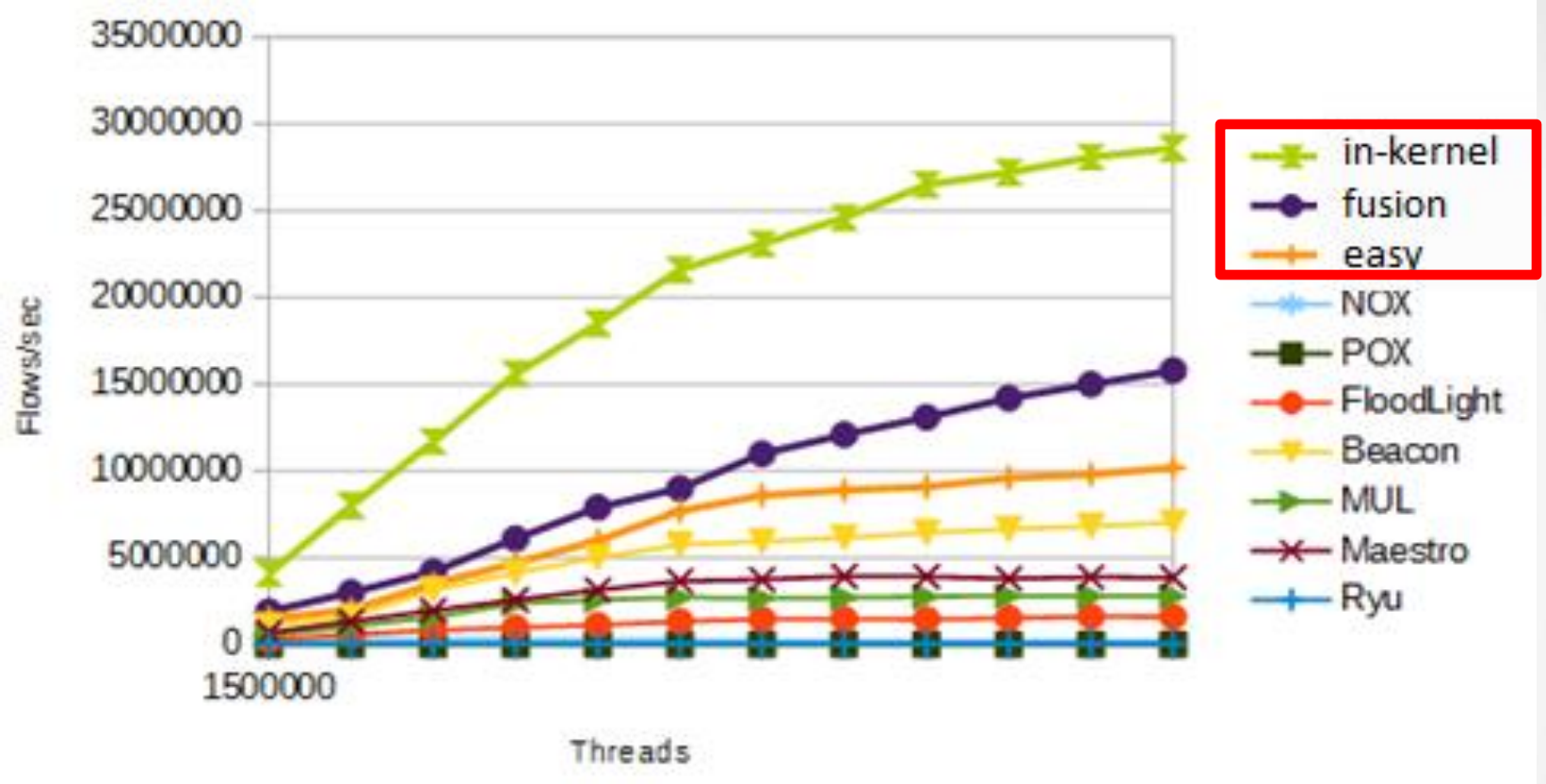
Проект с открытым исходным кодом

- Исходный код http://arccn.github.io/runos/
  - Apache, version 2.0
- Tutorial (Readme.md)
  - Как установить, запустить, написать свое первое приложение
- Виртуальная машина
  - Уже собранный контроллер
  - Средства для работы с OpenFlow
- Список рассылки
  - Google group **runos-ofc**

Программируемость

- Algorithmic policies (rule generation)
  - Arrangement of priorities of rules, combining of rules
  - LOAD, MATCH, READ abstractions
  - MAPLE based
- Client-friendly API using EDSL grammar (low level details are hidden inside the runtime – overloading, templates)
  - “pkt[eth src] == eth addr”
  - “if (ethsrc == A || ethdst == B) doA else doB”
  - “test((eth_src & “ff0…..0”) == “….”)”
  - *“modify(ip_dst >> “10.0.0.1”)”*
  - decision are “unicast()”, “broadcast()”, “drop()”
- Application composition (parallel and sequential composition)
  - dpi + (lb >> forwarding)

Производительность



- Производительность : **10 млн. потоков в секунду**
- Задержка: **55 мкс**

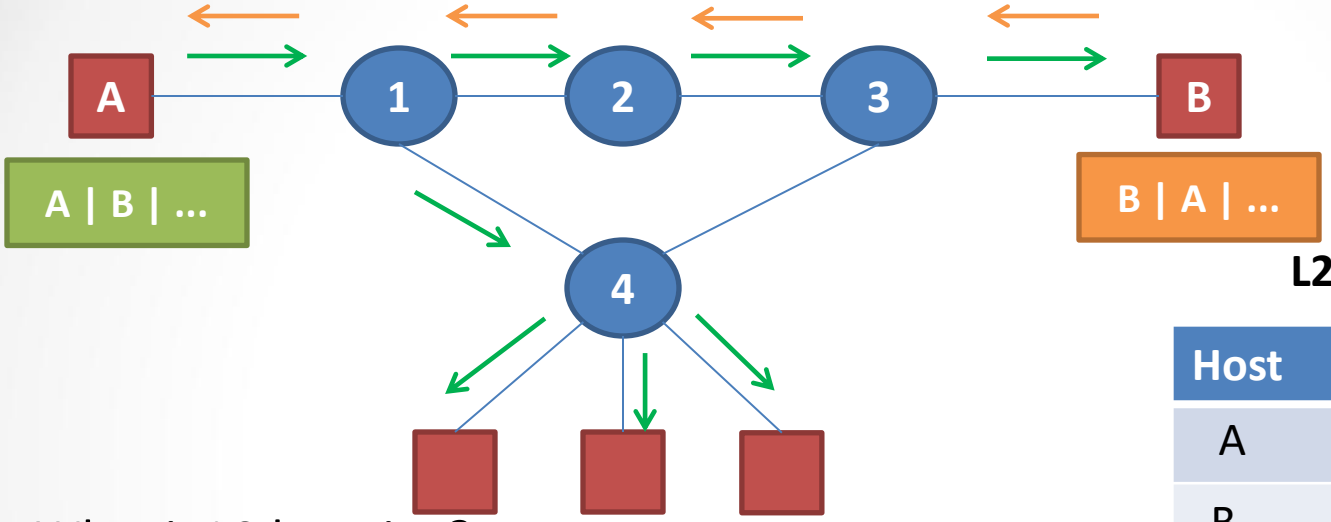
Open Source

- Два типа OpenSource проектов:
  - ради Идеи: “Free as in Freedom”
  - продаем свои компетенции, а не продукт
    - доработка под нужды заказчика,
    - продавать advanced версии и плагины (eg, приложения для Runos)
    - community вокруг (семинары, обучение)
- Важна лицензия (*): Apache, BSD или GPL, Eclipse, проприетарные лицензии, перелицензирование за деньги
- Угрозы:
  - Скопируют и будут продавать под другим названием (eg, runos-ng)
  - Будут дорабатывать своими силами (для компаний с большим R&D)

* http://www.slideshare.net/gerasiov/license-44646637

Часть IV: Разработка приложений для Runos контроллер

First application – L2 learning



L2 learning table

Host	Switch:port
A	1:1
B	3:2

- What is L2 learning?
  - L2 table – where particularly host resides (host <->sw:port)
- A->B. What should we do on sw1?
  - Learn and broadcast
- B->A. What should we do on sw3?
  - Learn and unicast
- **Advanced question: will it work for ping utilities? Ping 10.0.0.2 (assuming B has this IP)**
  - Yes, arp (broadcast), ip (icmp)

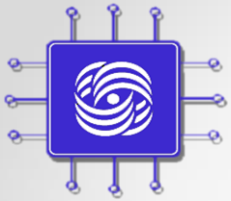
Host Databases

```
class HostsDatabase {
    boost::shared_mutex mutex;
    std::unordered_map<ethaddr, switch_and_port> db;

public:
    void learn(uint64_t dpid, uint32_t in_port, ethaddr mac)
    {
        LOG(INFO) << mac << " seen at " << dpid << ':' << in_port;
        {
            boost::unique_lock< boost::shared_mutex > lock(mutex);
            db[mac] = switch_and_port{dpid, in_port};
        }
    }

    boost::optional<switch_and_port> query(ethaddr mac)
    {
        boost::shared_lock< boost::shared_mutex > lock(mutex);

        auto it = db.find(mac);
        if (it != db.end())
            return it->second;
        else
            return boost::none;
    }
};
```

.insert(...)

L2 forwarding application

```
// Get required fields
ethaddr dst_mac = pkt.load(ofb_eth_dst);

db->learn(connection->dpid(),
          pkt.load(ofb_in_port),
          packet_cast<TraceablePacket>(pkt).watch(ofb_eth_src));

auto target = db->query(dst_mac);
// Forward
if (target) {
    flow->idle_timeout(60.0);
    flow->hard_timeout(30 * 60.0);

    auto route = topology->computeRoute(connection->dpid(),
                                        target->dpid);

    if (route.size() > 0) {
        flow->unicast(route[0].port);
    } else {
        flow->idle_timeout(0.0);
        LOG(WARNING) << "Path from " << connection->dpid()
            << " to " << target->dpid << " not found";
    }

    return PacketMissAction::Continue;
} else {
    flow->broadcast();
    return PacketMissAction::Continue;
}
```



Пример работы с Runos

- Инструментарий
  - Runos
  - Mininet
- Перейти на виртуальную машину

Часть V: Распределенный уровень управления

Высокая доступность (High Availability, HA)

ЦЕНТР ПРИКЛАДНЫХ ИССЛЕДОВАНИЙ КОМПЬЮТЕРНЫХ СЕТЕЙ

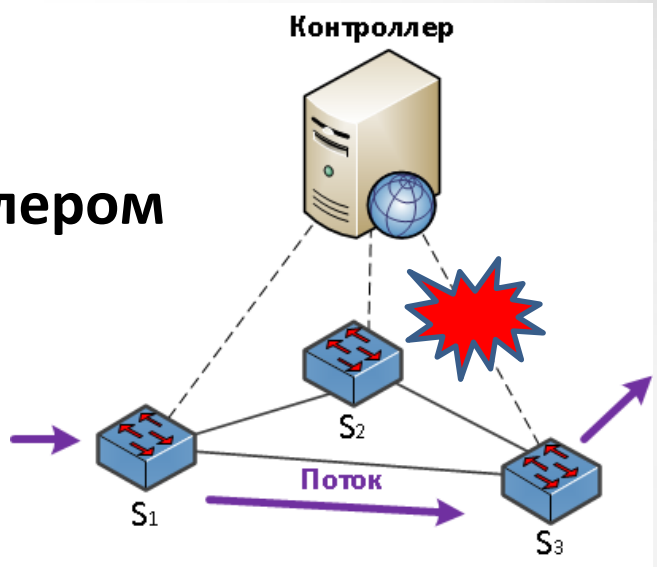
- Сеть работает в режиме 365/24/7.
- Платформа управления ПКС должна работать непрерывно.
- Цель обеспечения высокой готовности – поддержание непрерывной работоспособности платформы управления, сетевых приложений.
- Причины простоя: обслуживание, программные и аппаратные ошибки, отказы оборудования, атаки, отключение электроэнергии, аварии.

Коэффициент готовности, %	Время простоя за год
99,999	5 минут
99,99	52 минуты
99,9	8,7 часов
99	3,7 дней

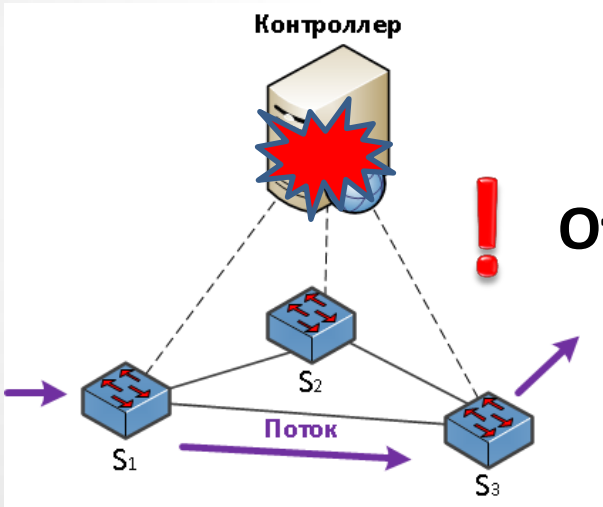
Угрозы

ЦЕНТР ПРИКЛАДНЫХ ИССЛЕДОВАНИЙ КОМПЬЮТЕРНЫХ СЕТЕЙ

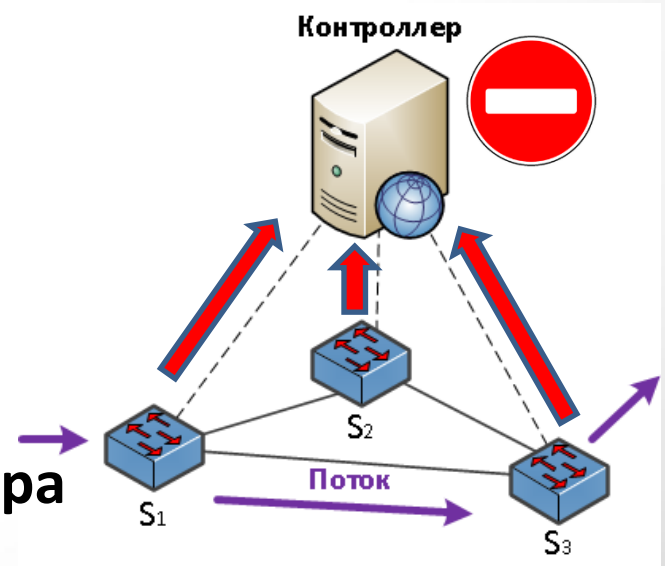
Потеря соединения коммутатора с контроллером



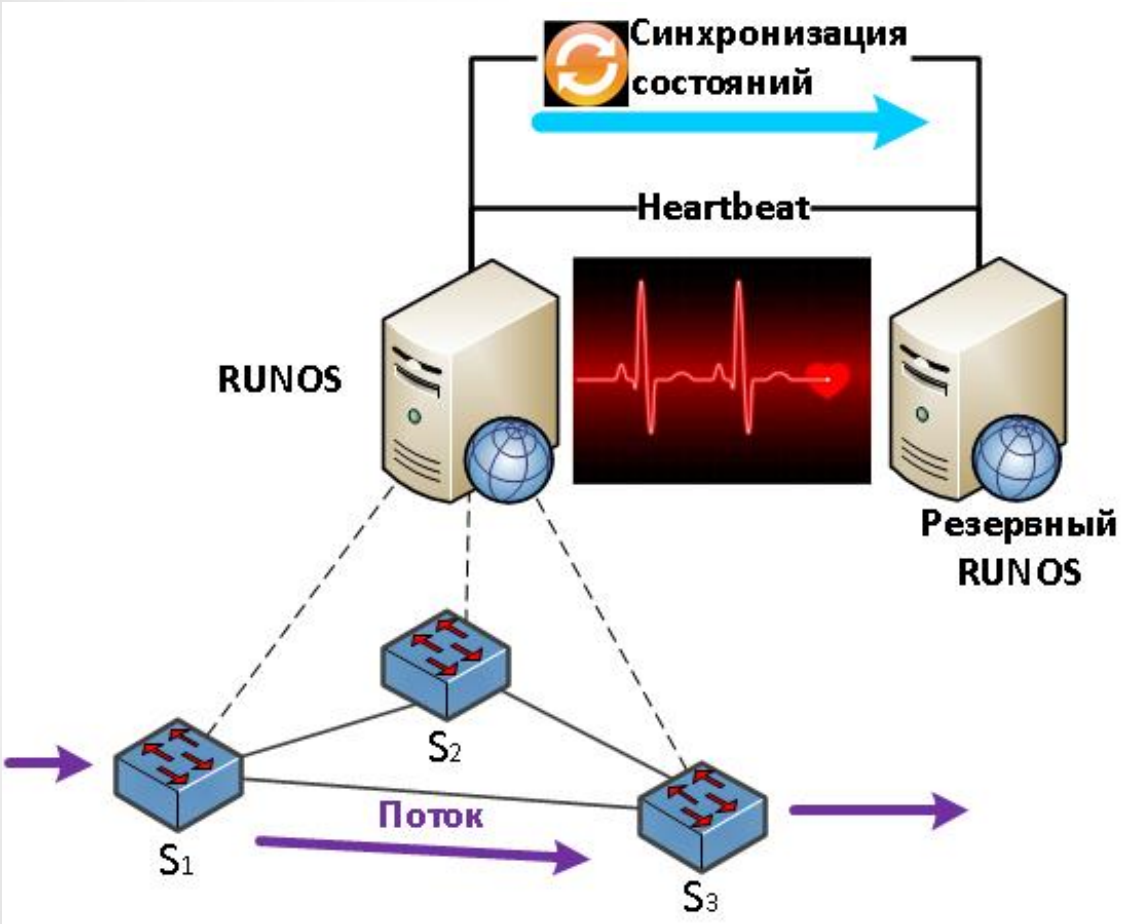
Отказ контроллера



Перегрузка контроллера



Стратегии резервирования /1



Active/Standby (Passive) **стратегии**:

- **Холодное**

[без синхронизации]

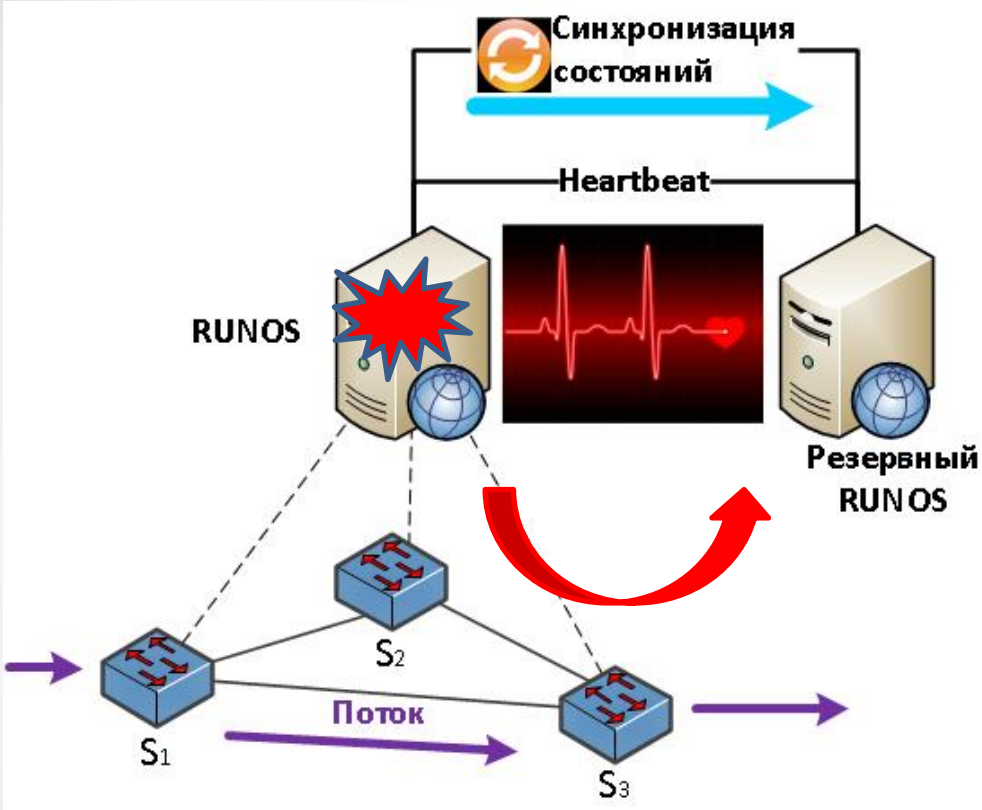
- **Теплое**

[периодическая синхронизация]

- **Горячее**

[постоянная синхронизация]

Восстановление после отказа контроллера для случая горячего резервирования RUNOS



Особенности решения:

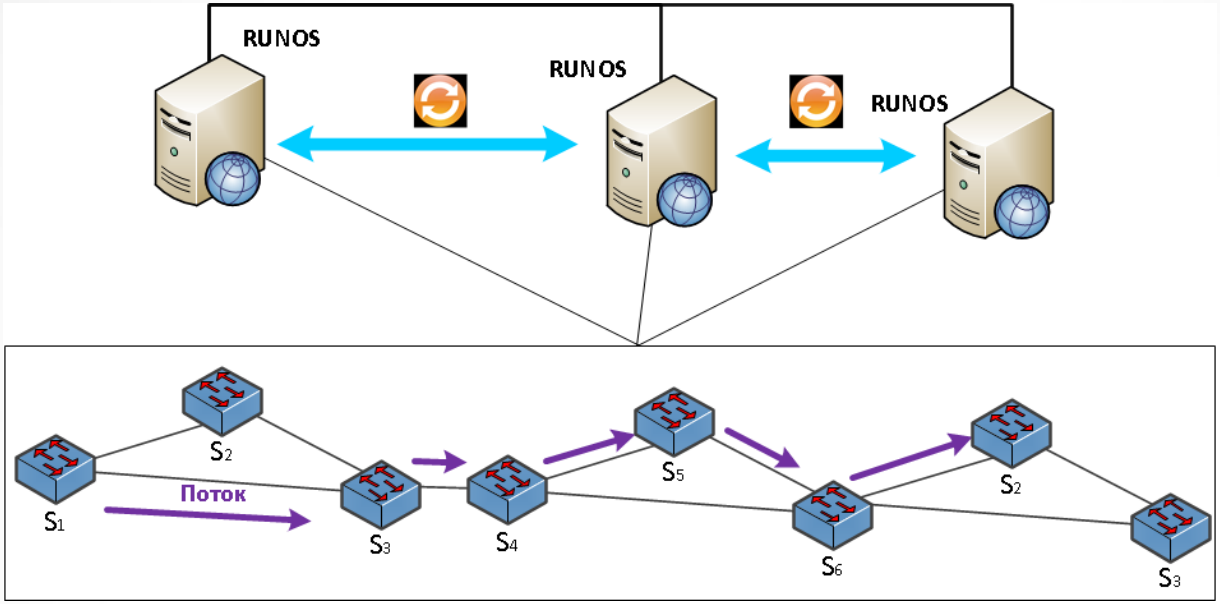
- OpenFlow ≥ 1.0
- Корпоративные сети.
- Не масштабируется
- Не полная утилизация вычислительных ресурсов

\+ Одиночный отказ контроллера

\- Потеря соединения коммутатора с контроллером

\- Перегрузка контроллера

Стратегии резервирования /2

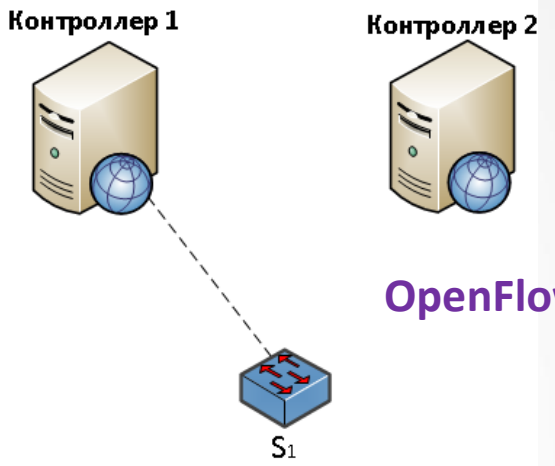


- Стратегии резервирования **Active/Active**
  - Ассиметричная
  - **Симметричная**
- Высокая сложность [Требуется координация контроллеров, поддержка глобального состояния]
- Высокая доступность [минимальное время простоя]
- Высокая утилизация вычислительных ресурсов

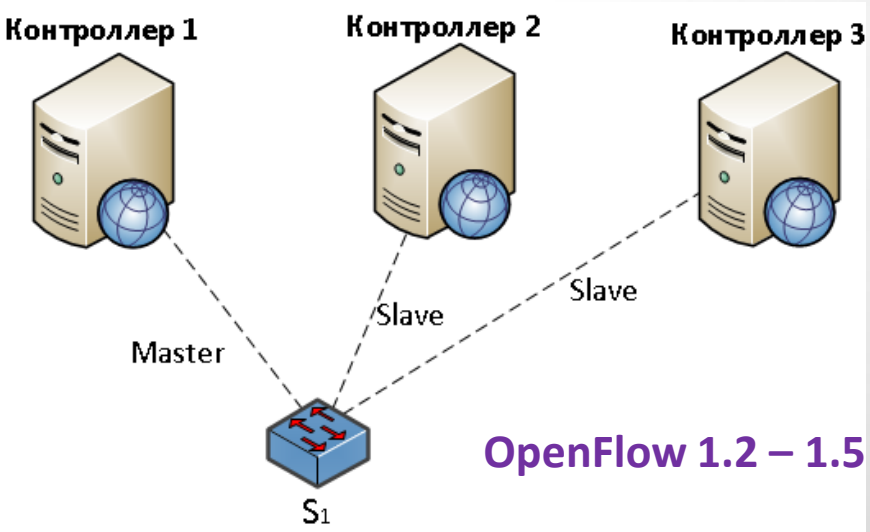
Возможности протокола OpenFlow

ЦЕНТР ПРИКЛАДНЫХ ИССЛЕДОВАНИЙ КОМПЬЮТЕРНЫХ СЕТЕЙ

- Протокол OpenFlow 1.2:
  - Множество контроллеров
  - Механизм ролей
  - **Роли:** Master, Slave, Equal
  - **По умолчанию:** контроллер находится в роли Equal для коммутаторов.
  - **Смена роли:** OFPT_ROLE_REQUEST
  - **Распределение ролей:** возложено на контроллеры.

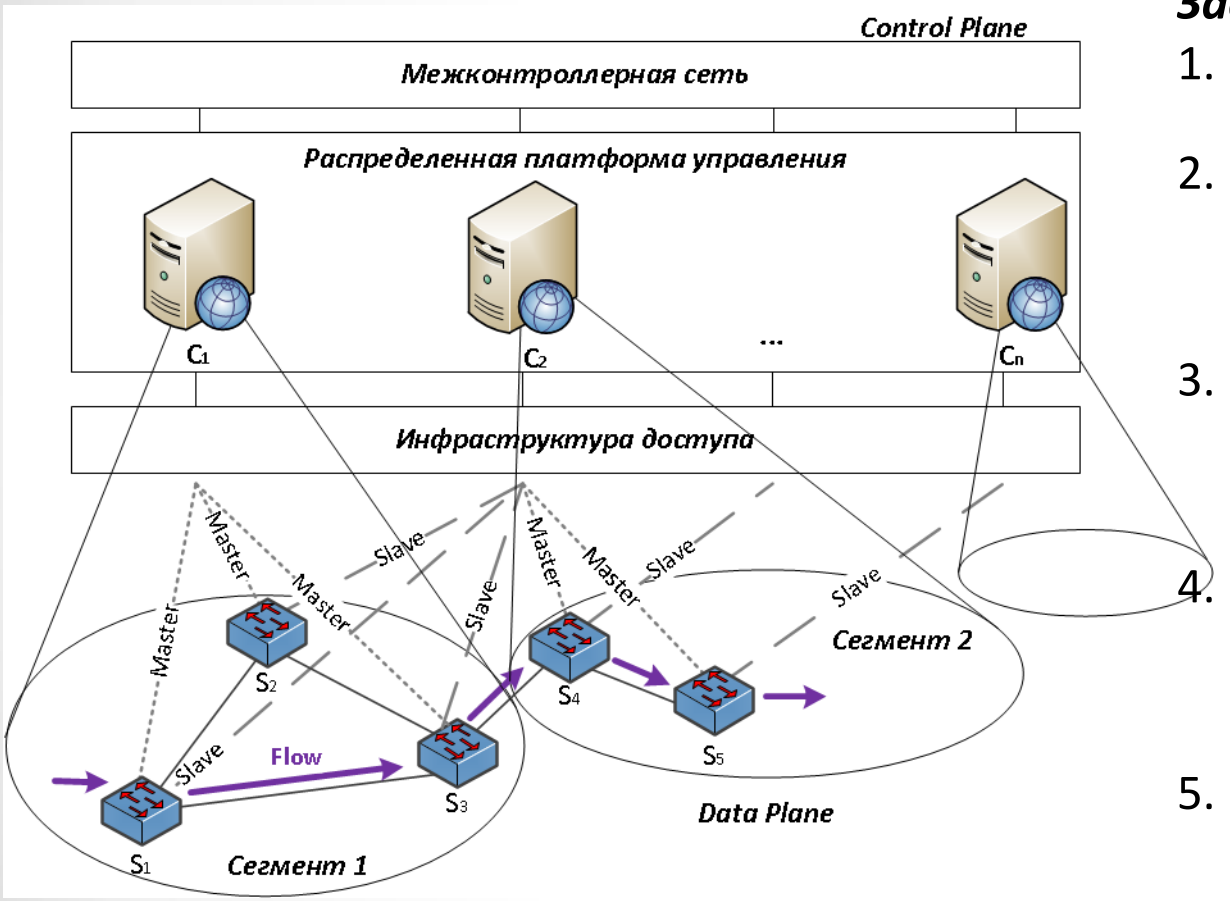


OpenFlow 1.0, 1.1



OpenFlow 1.2 – 1.5

Модель платформы управления



Задачи:

1. Как синхронизовать контроллеры?
2. Как определить начальное распределение коммутаторов по контроллерам?
3. Как перераспределить управление коммутаторами при одиночном отказе контроллера?
4. Как восстановить управление коммутатором при потере соединения с контроллером?
5. Как предотвратить перегрузку контроллера в платформе управления?

Предположения модели

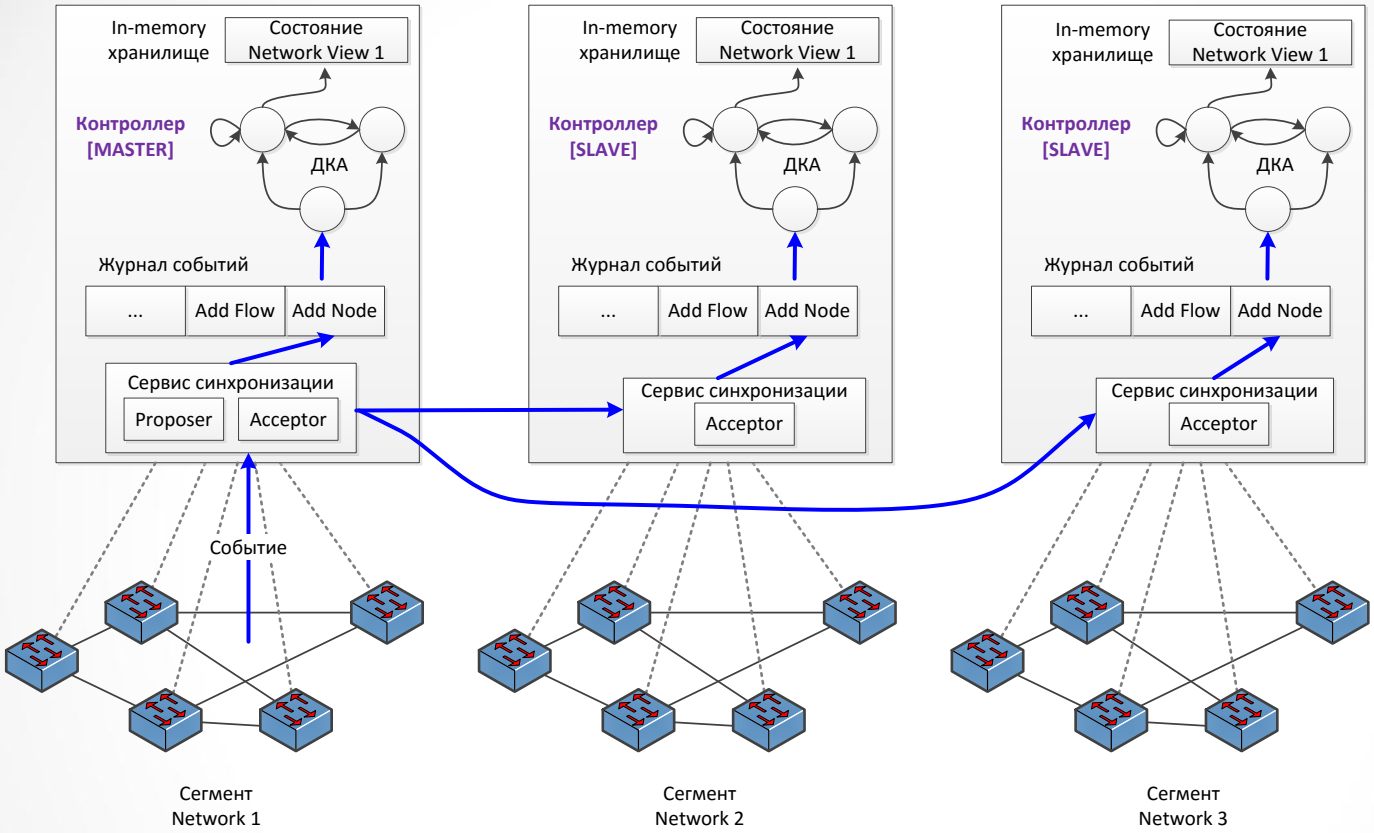
Предположения:

- Все контроллеры имеют одинаковую производительность.
- Контроллер устанавливает правила только на те коммутаторы, для которых он является Master контроллером.
- Инфраструктура доступа обеспечивает связь коммутаторов с каждым контроллером.

Условие корректности решения:

- В каждый момент времени для каждого коммутатора в сети должен быть определен только один Master контроллер.

Задача 1: Синхронизация контроллеров

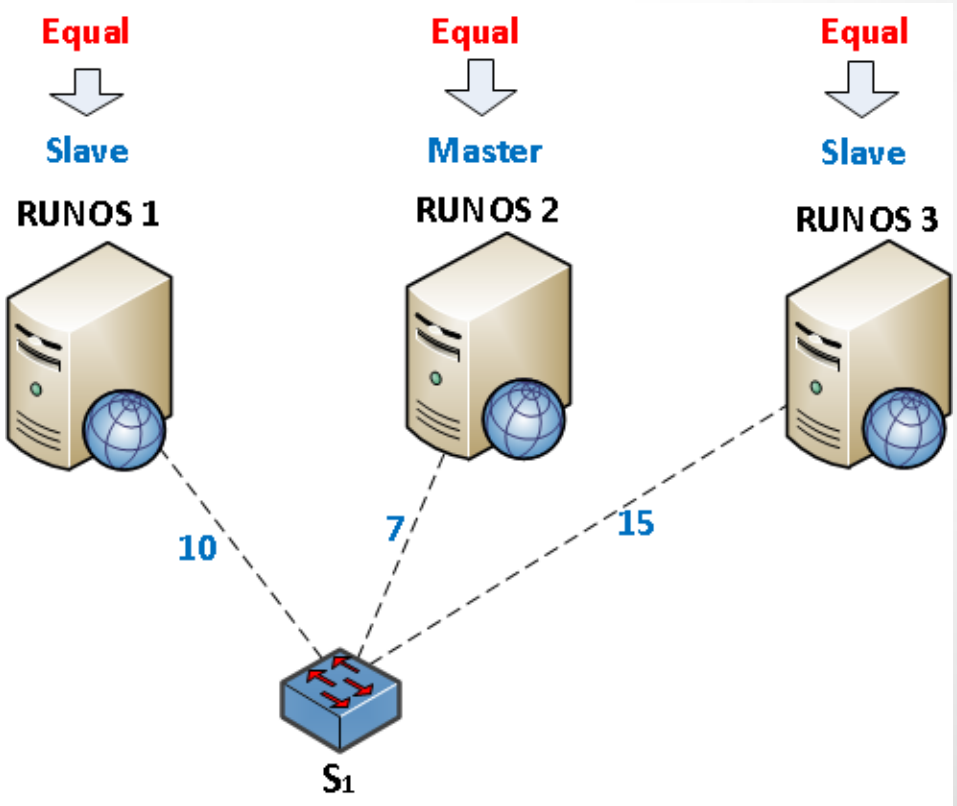


- На основе Multi-Paxos
- Обеспечивает одновременное одинаковое выполнение команд контроллеров
- Команды: изменение Network View, установление новых потоков (чтение NV), изменение ролей контроллеров.

Задача 2: Начальное распределение коммутаторов по контроллерам

ЦЕНТР ПРИКЛАДНЫХ ИССЛЕДОВАНИЙ КОМПЬЮТЕРНЫХ СЕТЕЙ

- **Критерий определения Master-контроллера:**
  - Задержка от коммутатора до контроллера, как мера близости.
- **Ограничение:**
  на количество коммутаторов в сегменте контроллера.
- **Решение:**
  Жадный алгоритм по значению задержки от коммутатора до контроллеров.



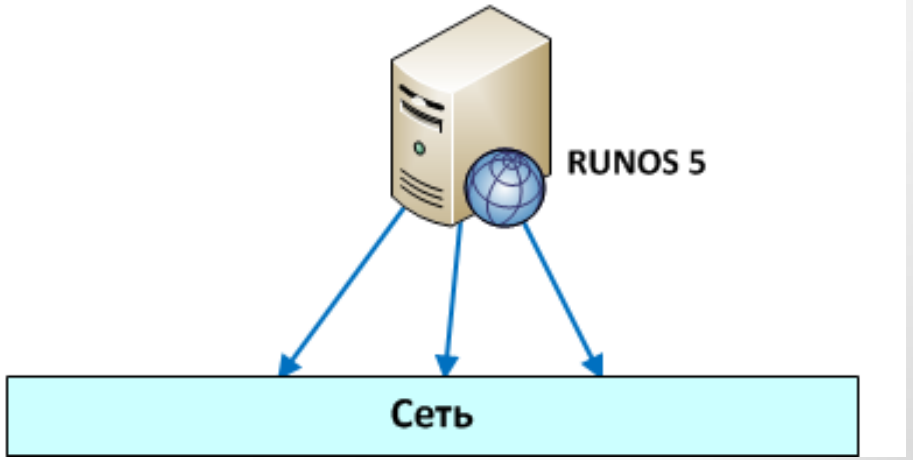
Задача 3: Выработка и представление сценариев восстановления

ЦЕНТР ПРИКЛАДНЫХ ИССЛЕДОВАНИЙ КОМПЬЮТЕРНЫХ СЕТЕЙ

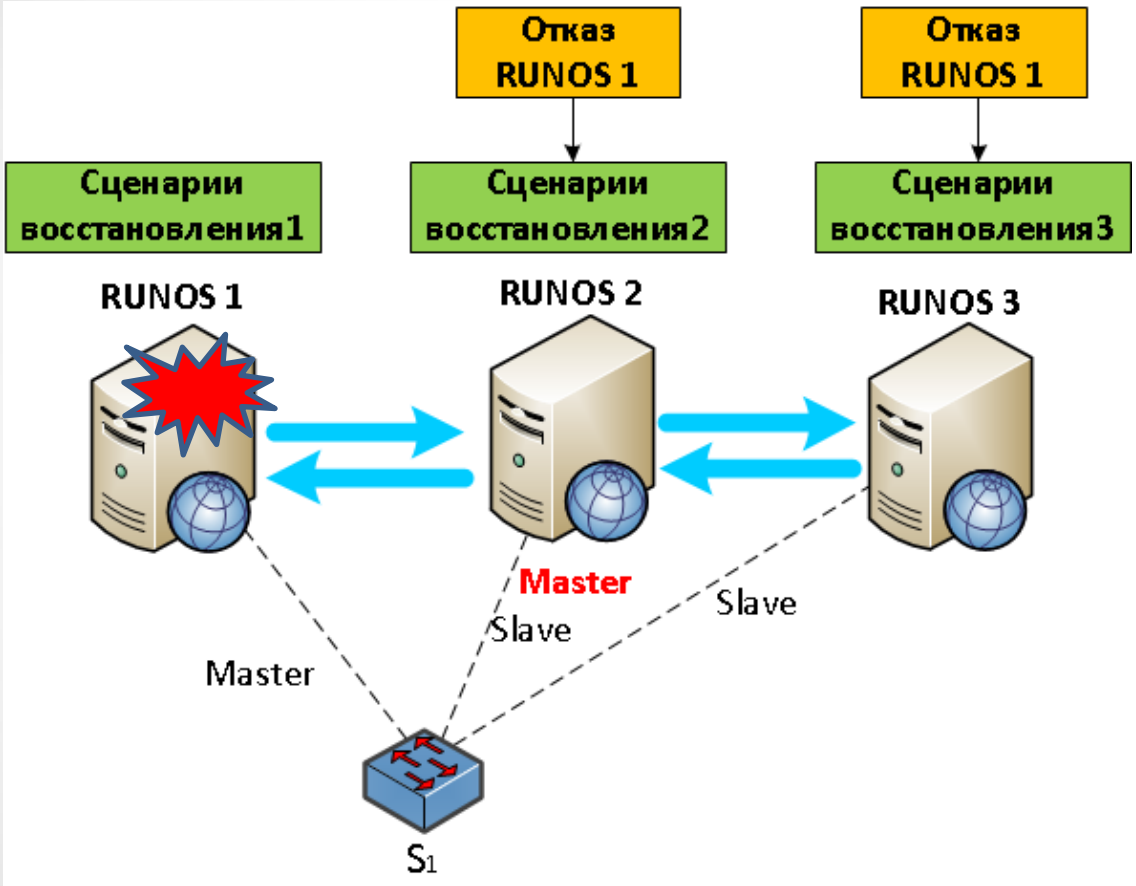
- Проактивная разработка сценариев восстановления
- Критерий выбора нового распределения: задержка от коммутатора до контроллера
- Алгоритм Балаша.
- **Результат:** перечень сценариев для каждого контроллера платформы управления.

Сценарии восстановления

Отказ контроллера	Перечень коммутаторов, для которых контроллер должен стать Master-ом
RUNOS 01	S3,S4
RUNOS 02	S1, S5
…	…
RUNOS N	S k

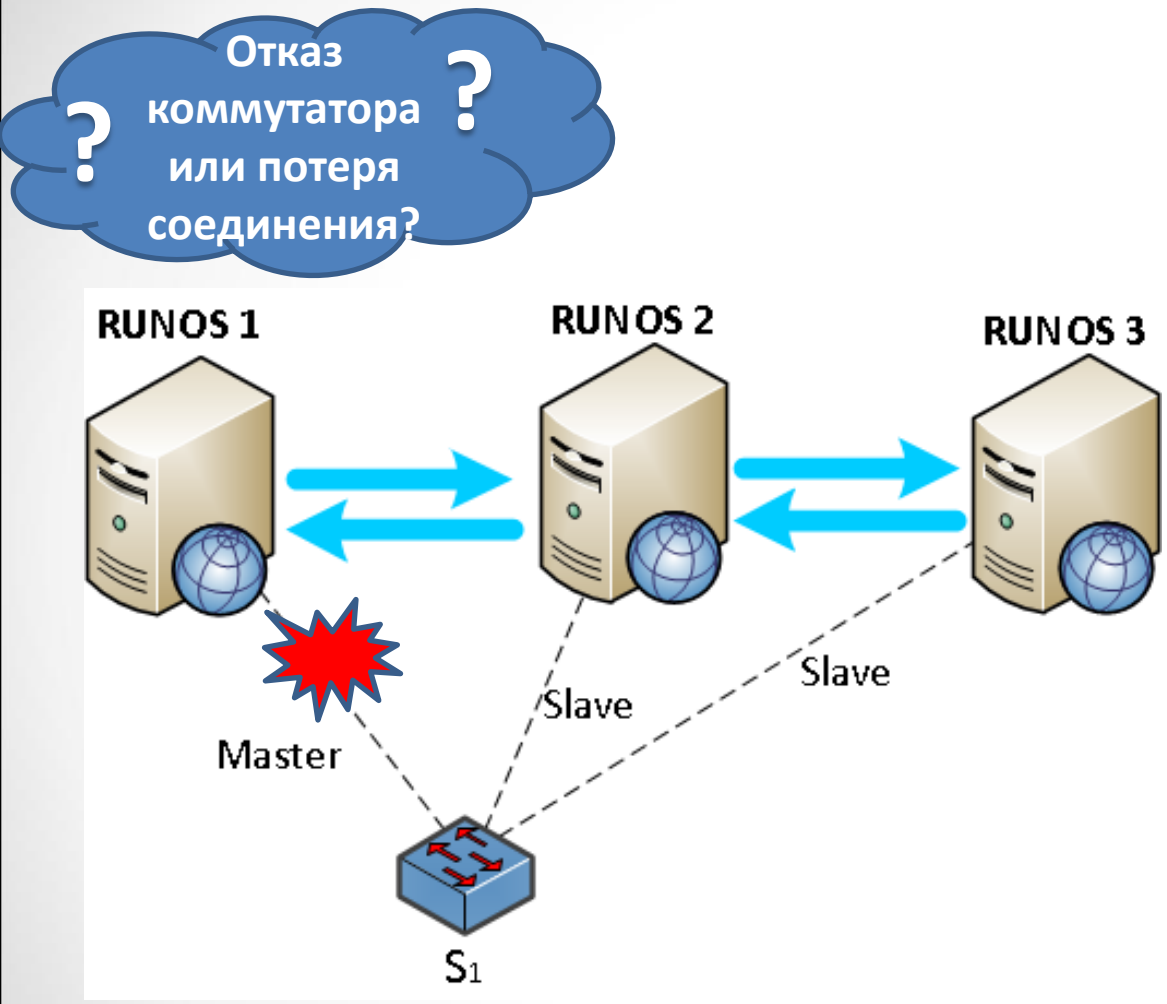


Задача 3: Использование сценария для восстановления ПУ после отказа контроллера



1. Оставшимися контроллерами фиксируется факт отказа контроллера и CID.
2. Каждый контроллер выбирает сценарий восстановления, соответствующий отказу контроллера CID.
3. В соответствии со сценарием каждый контроллер изменяет свою роль на коммутаторах.
4. Каждый контроллера информирует остальные контроллеры о завершении выполнения сценария восстановления.

Задача 4: Восстановление при потере соединения коммутатора с контроллером



1. Фиксируется отключение коммутатора Master-контроллером.
2. Инициируется проверка присутствия коммутатора в сети.
3. Осуществляется измерении задержки контроллерами, поддерживающими связь с коммутатором.
4. Master-контроллером становится контроллер с минимальной задержкой до коммутатора.
5. Новый Master-контроллер устанавливает свою роль на коммутаторе.

Задача 5: Предотвращение перегрузки контроллера платформы управления

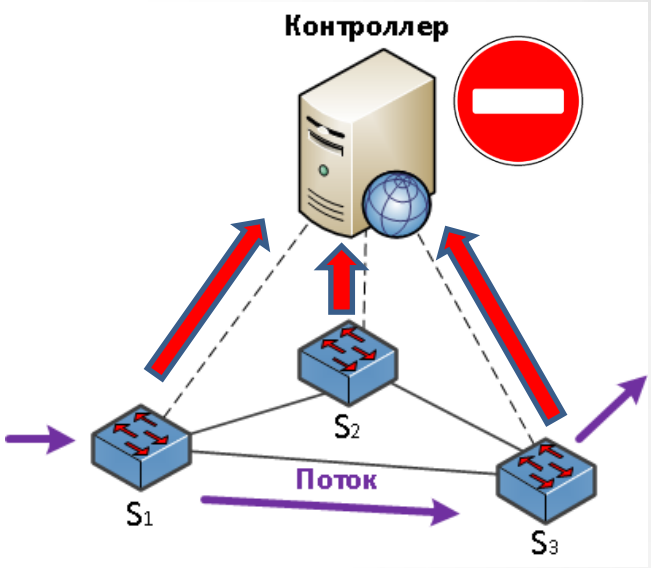
ЦЕНТР ПРИКЛАДНЫХ ИССЛЕДОВАНИЙ КОМПЬЮТЕРНЫХ СЕТЕЙ

Для обнаружения перегрузки:

- Устанавливаются пороговые значения параметров загрузки контроллеров.
- Осуществляется постоянный мониторинг загрузки контроллеров

[количество коммутаторов, количество packet-in запросов в секунду, CPU, память]

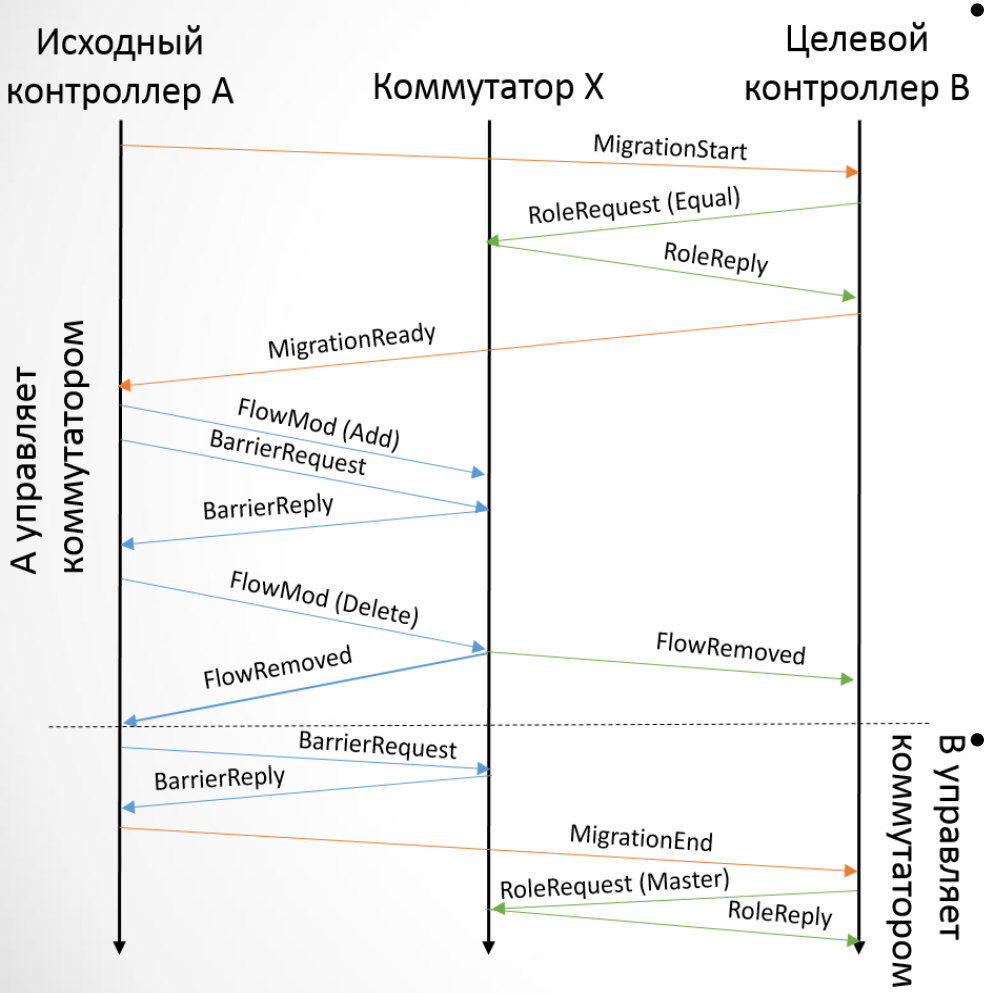
- Факт перегрузки фиксируется при обнаружении превышения порогового значения.



Задача перераспределения коммутаторов по контроллерам: Необходимо перераспределить коммутаторы так, чтобы:

1. Нагрузка на каждый из контроллеров не превышала его производительности
2. Использовалось минимальное количество контроллеров.
3. При перестроении управления сетью было проведено минимальное количество передач управления коммутаторами.

Задача 5: Передача управления коммутатором



- **Свойство живучести:** В любой момент времени для коммутатора существует контроллер, работающий в режиме *Master.* Для любой асинхронной команды, контроллер, который ее отправил, остается активным до тех пор, пока коммутатор не закончит ее обработку.

- **Свойство безопасности:** Ровно один контроллер обрабатывает каждое асинхронное сообщение от коммутатора.

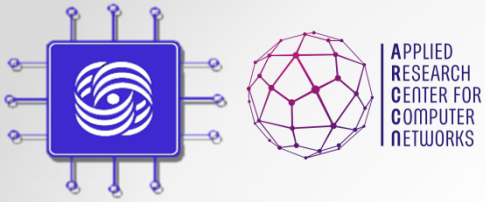
Задача 5: Распределение коммутаторов по группам

- **Входные данные:**
  - Топология сети
  - Количество новых потоков с каждого коммутатора
  - Максимально допустимая нагрузка на контроллер *P*
- **Алгоритм** разбиения графа на компоненты связности, чтобы суммарная нагрузка компоненты связности не превышала *P.*
- **Выходные данные:**
  - Матрица, определяющая Master-контроллеры для коммутаторов и сегменты управления для каждого контроллера.

Задача 5: Распределение групп коммутаторов по контроллерам

ЦЕНТР ПРИКЛАДНЫХ ИССЛЕДОВАНИЙ КОМПЬЮТЕРНЫХ СЕТЕЙ

- **Входные данные:**
  - Текущая матрица распределения управления коммутаторами по контроллерам.
  - Желаемая матрица распределения управления коммутаторами.
- **Алгоритм:** жадный алгоритм, минимизирующий количество передач управления коммутаторами между контроллерами платформы управления.
- **Выходные данные:**
  - Список операций по передаче управления отдельными коммутаторами.

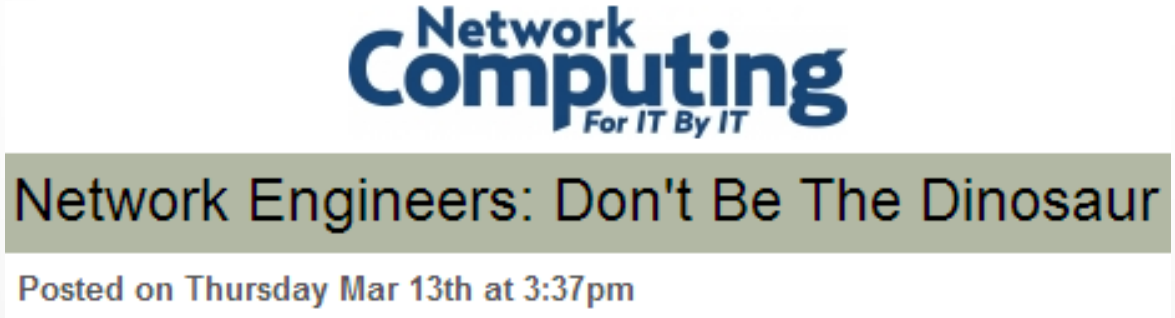


Quiz 2

- На другом слайде

Заключение

- OpenFlow Tutorial
  - гуглится
- Runos проект предназначен для упрощения разработки новых приложений
  - В открытом доступе arccn.github.io/runos



http://arccn.ru/
ashalimov@lvk.cs.msu.su
@alex_shali
@arccnnews