Skoltech
Skolkovo Institute of Science and Technology

Lomonosov Moscow State University

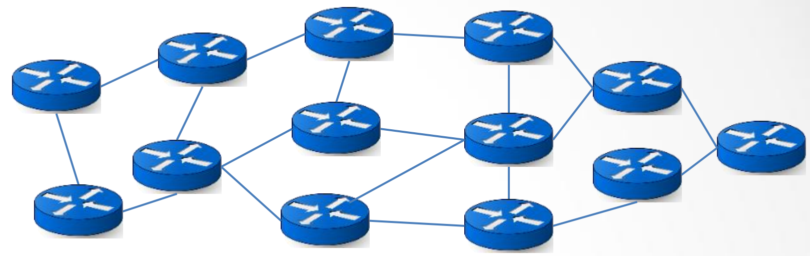
SDN&NFV: Software-Defined Networks (SDN)

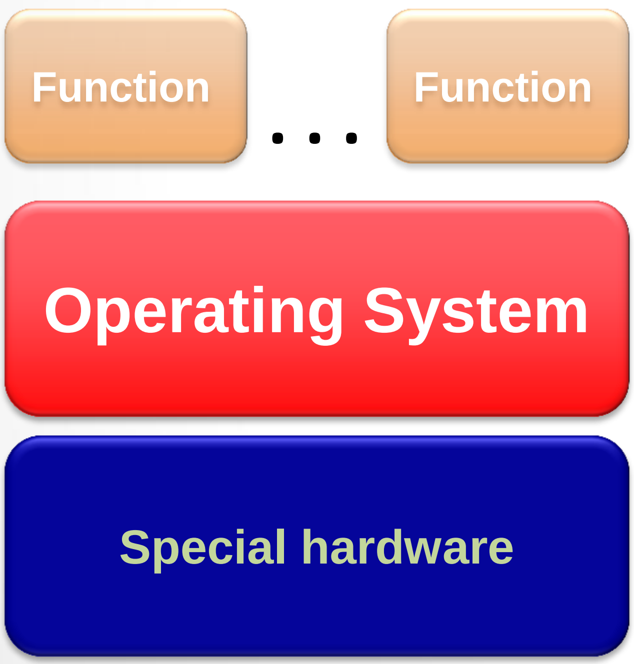
Advanced Computer Networks

Vasily Pashkov
pashkov@lvk.cs.msu.su

Part I: SDN

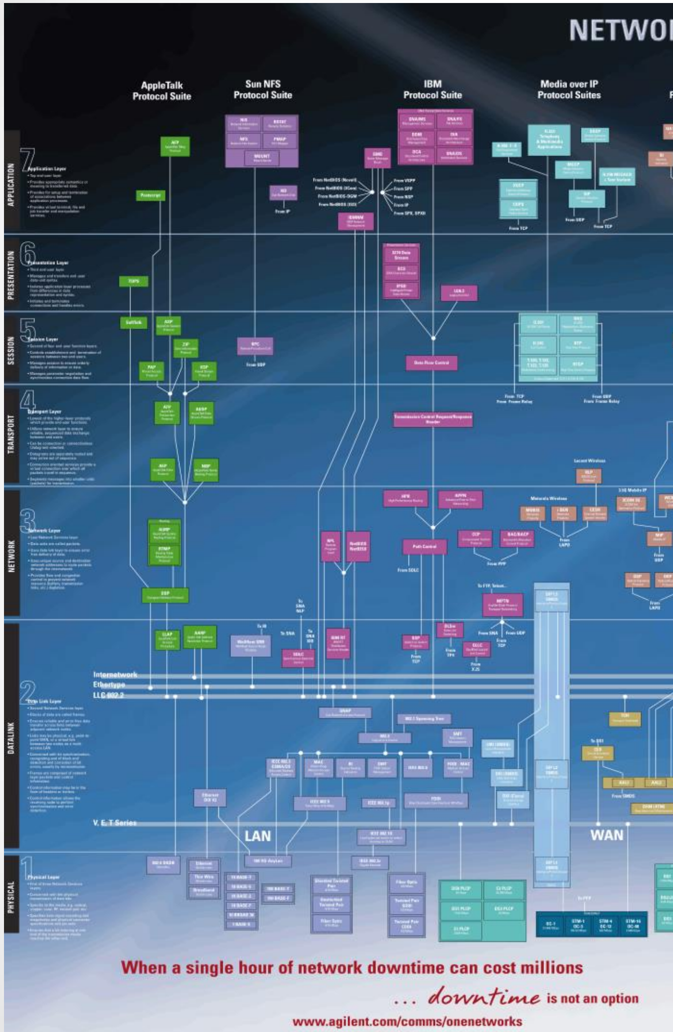
Network problems

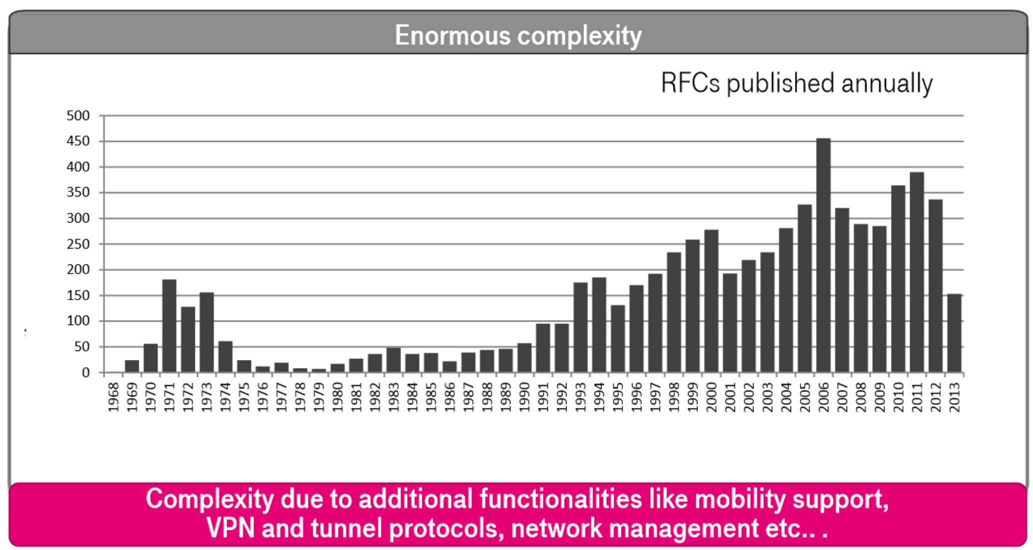




- Manufacturer dependent
- Errors in network protocol implementations
- Millions of Proprietary Closed Lines (6000+ RFC)
- High cost of equipment
- High cost of operation
- The difficulty of managing large networks
- Debugging complexity
- “Closed” equipment and software
- The difficulty of introducing new ideas
- Inefficiency in the use of hardware resources, energy inefficiency

Constantly increasing difficulty





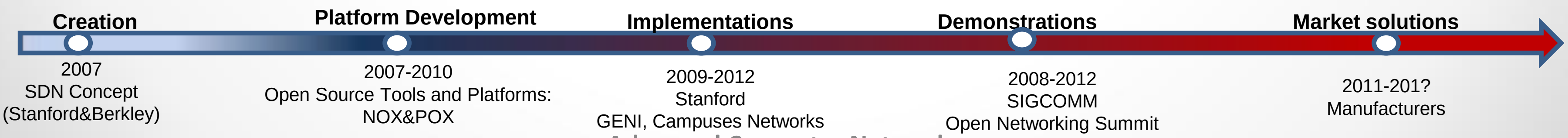
Source: http://www.telegeography.com/products/ip_transit/index.php; http://www.ietf.org/

SDN principles

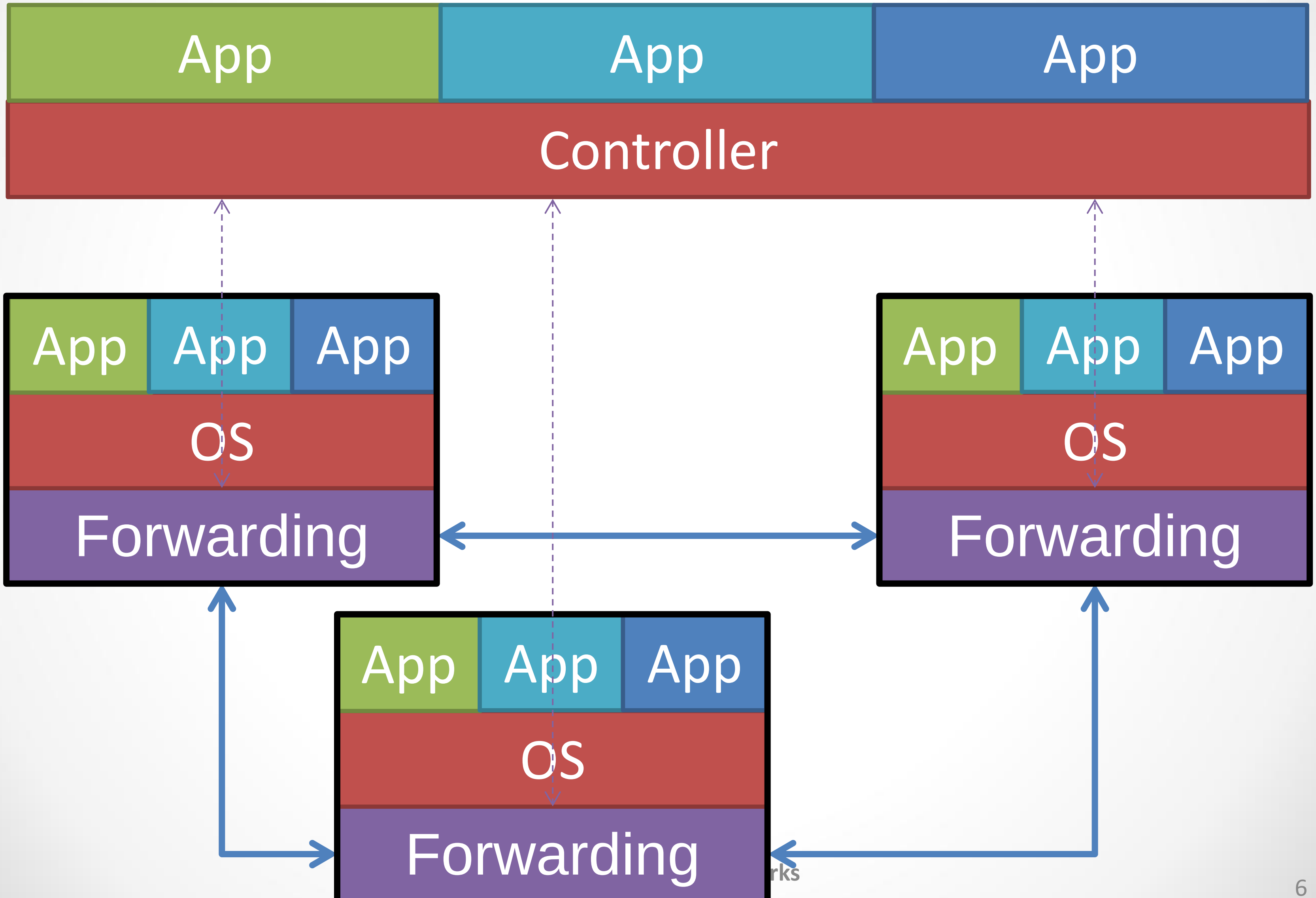
Software **D**efined **N**etworking (**SDN**)

1. Physically separate the network control plane from the data plane.
2. Logically centralized network management.
3. Open interface between the control plane and the data plane.

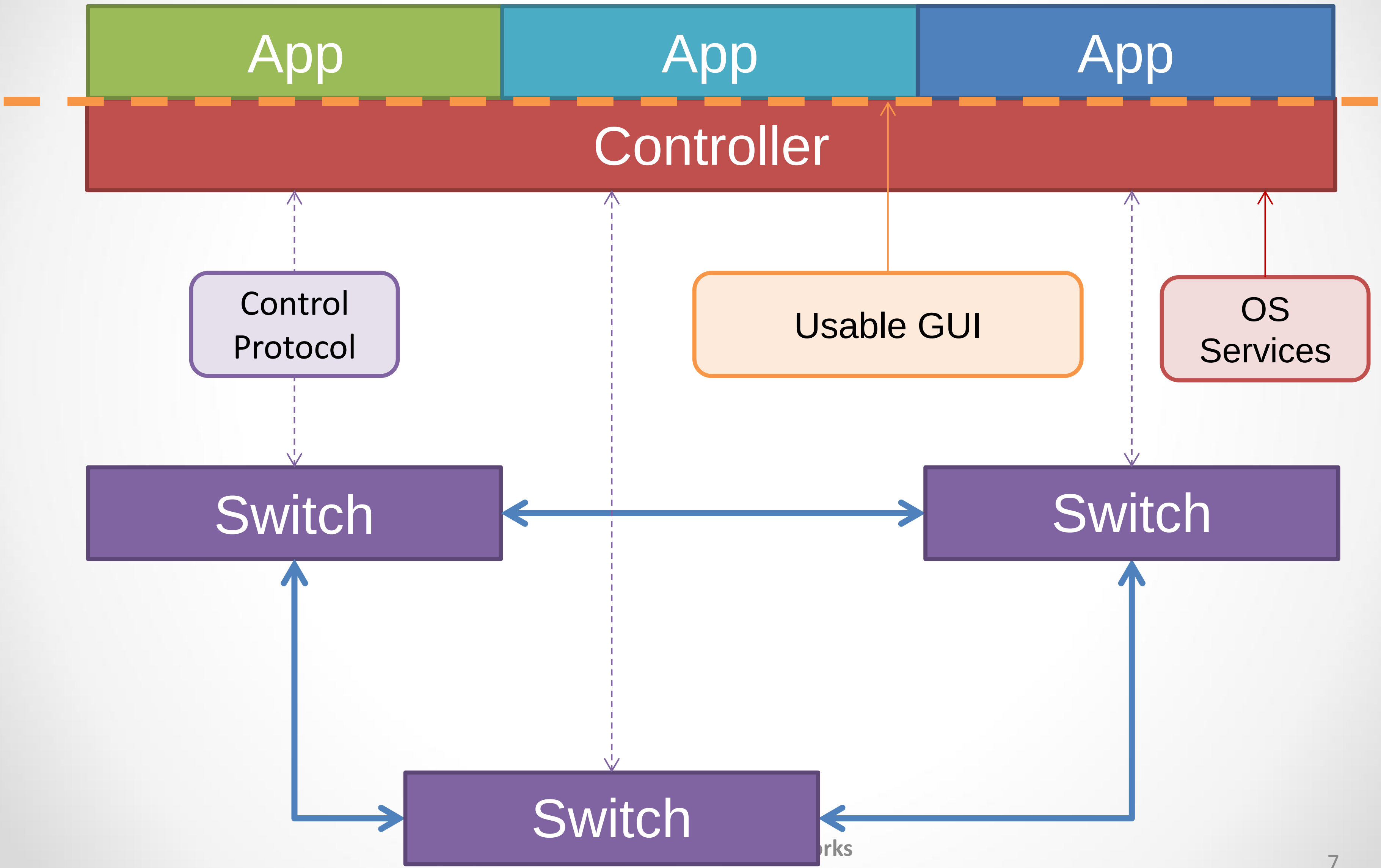
- New tools and features;
- Ease of administration;
- Openness to innovation and experimentation;
- IT market revolution



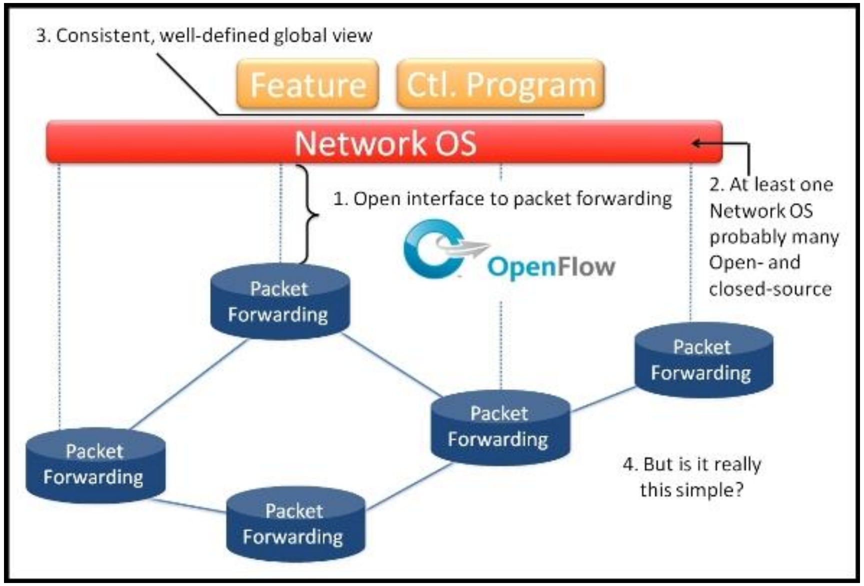
Forward to SDN



SDN Architecture

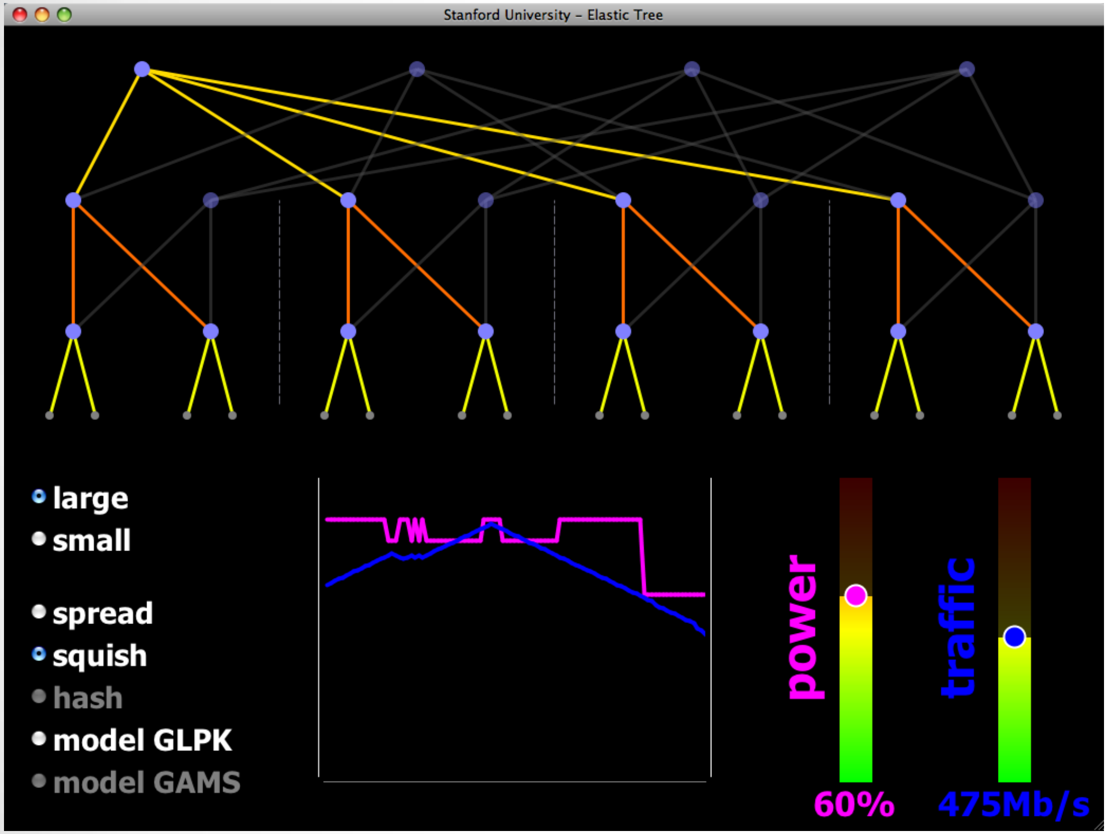


SDN Advantages



- **Management flexibility**
- **Cheaper equipment (reduced CAPEX)**
- **Facilitate network management (OPEX reduction)**
- **Programmability, openness, innovation**

SDN Use Case



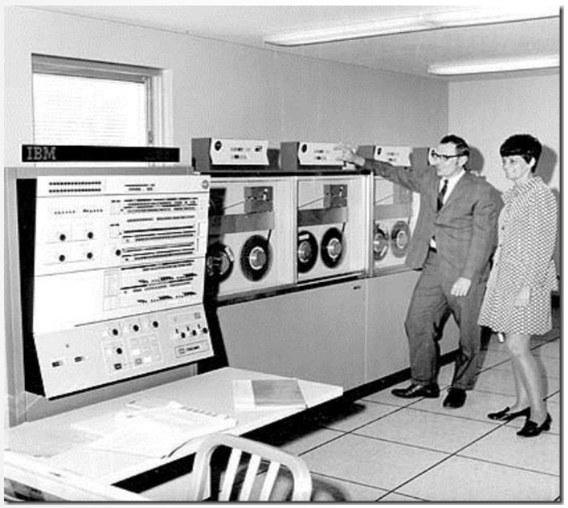
Decrease in power consumption in the data centers:

- Disabling Unused Switches and Channels Based on Collected Network Information
- ElasticTree (Stanford): up to 60% less energy
- Google application

Abstractions in IT

Slowly developing, closed, expensive system.
Small market

Rapid innovations
Open interfaces
Large market



Applications

Open Interface

Operating System

Windows

iOS

Open Interface

Microprocessors

intel

AMD

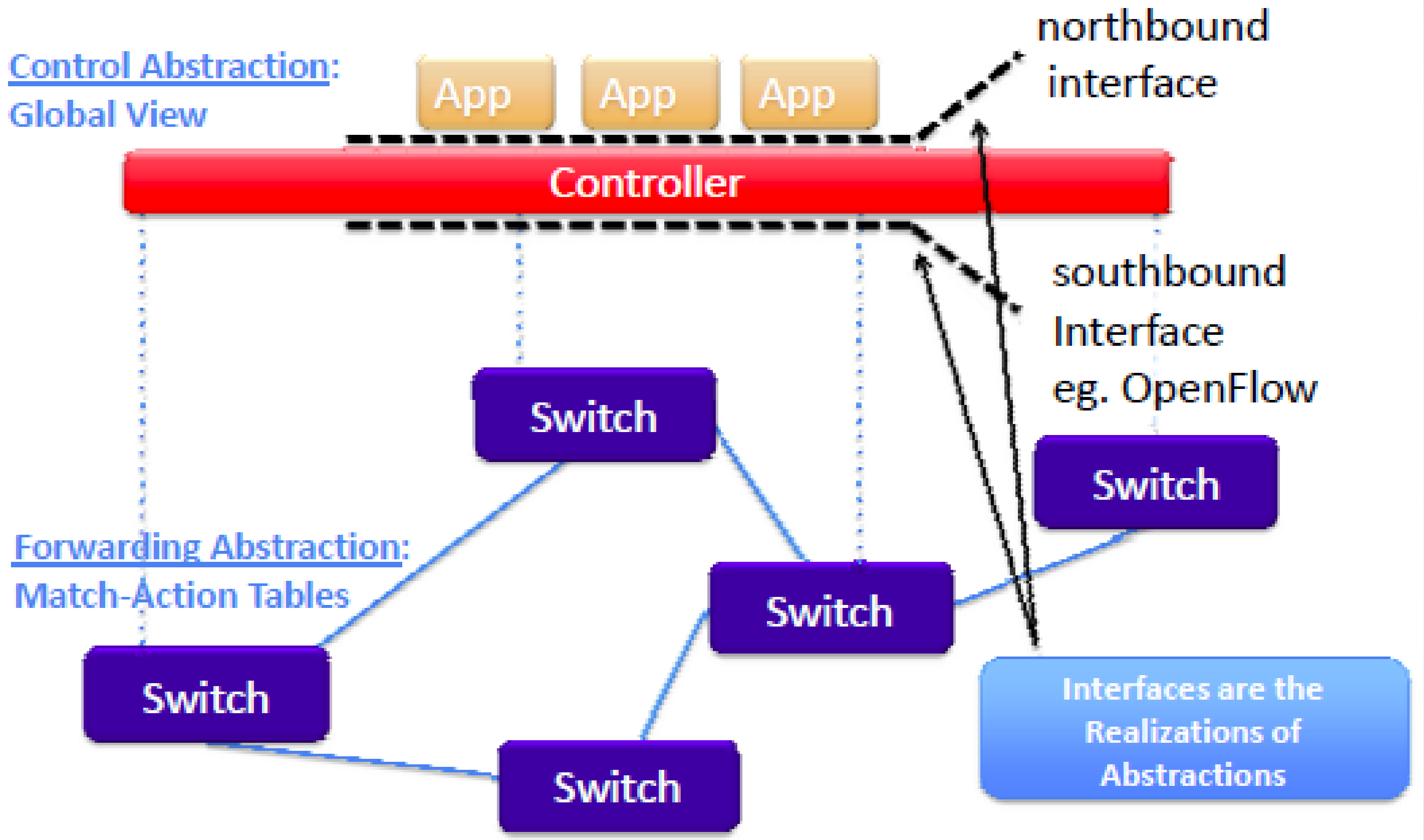
QUALCOMM

Specialized Programs

Specialized Operating System

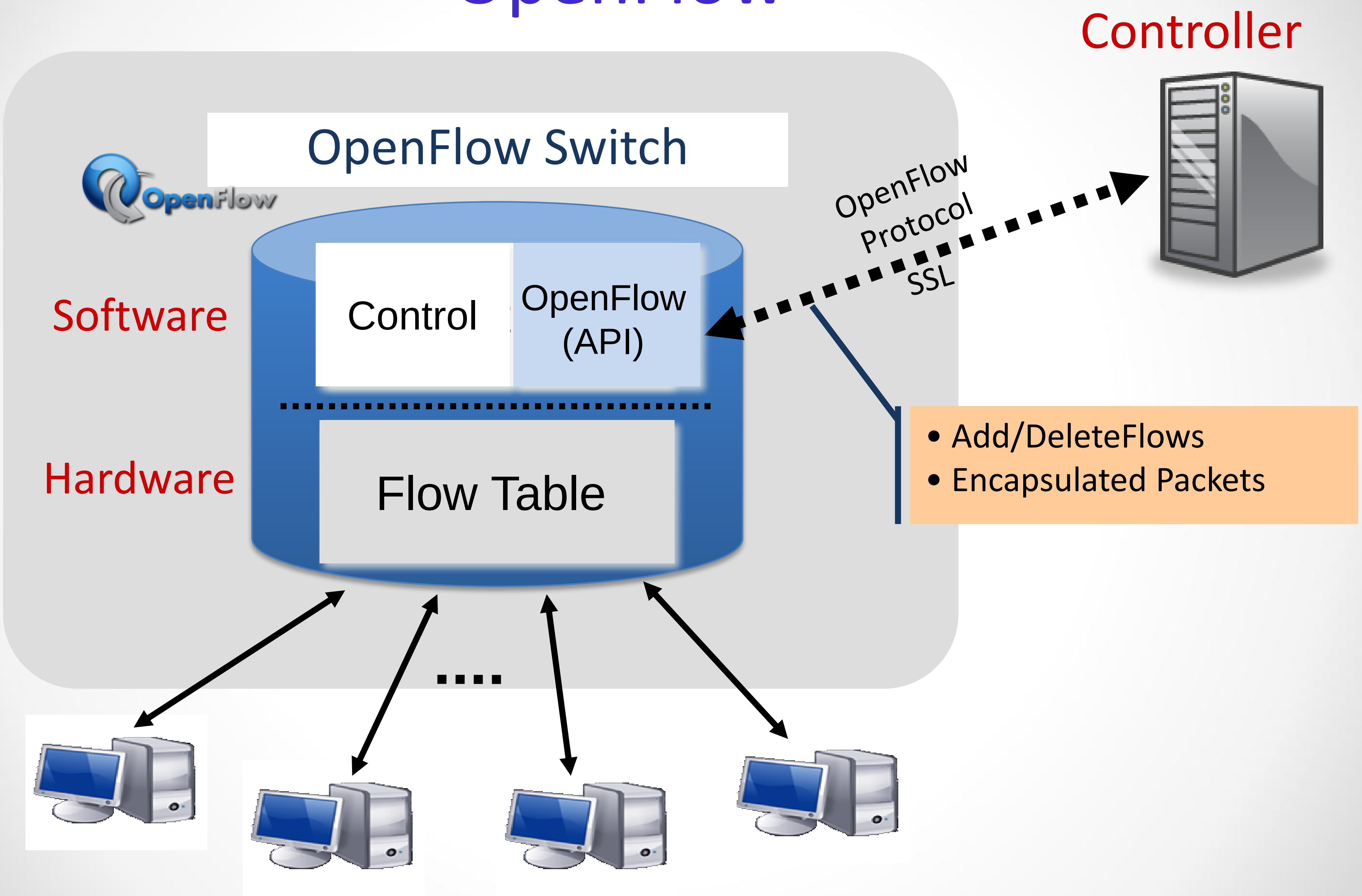
Specialized Hardware

Abstractions в SDN?



Part II: OpenFlow

OpenFlow


Controller
OpenFlow Switch
OpenFlow
Software
Hardware
Control
OpenFlow (API)
Flow Table
OpenFlow Protocol
SSL
• Add/DeleteFlows
• Encapsulated Packets
....

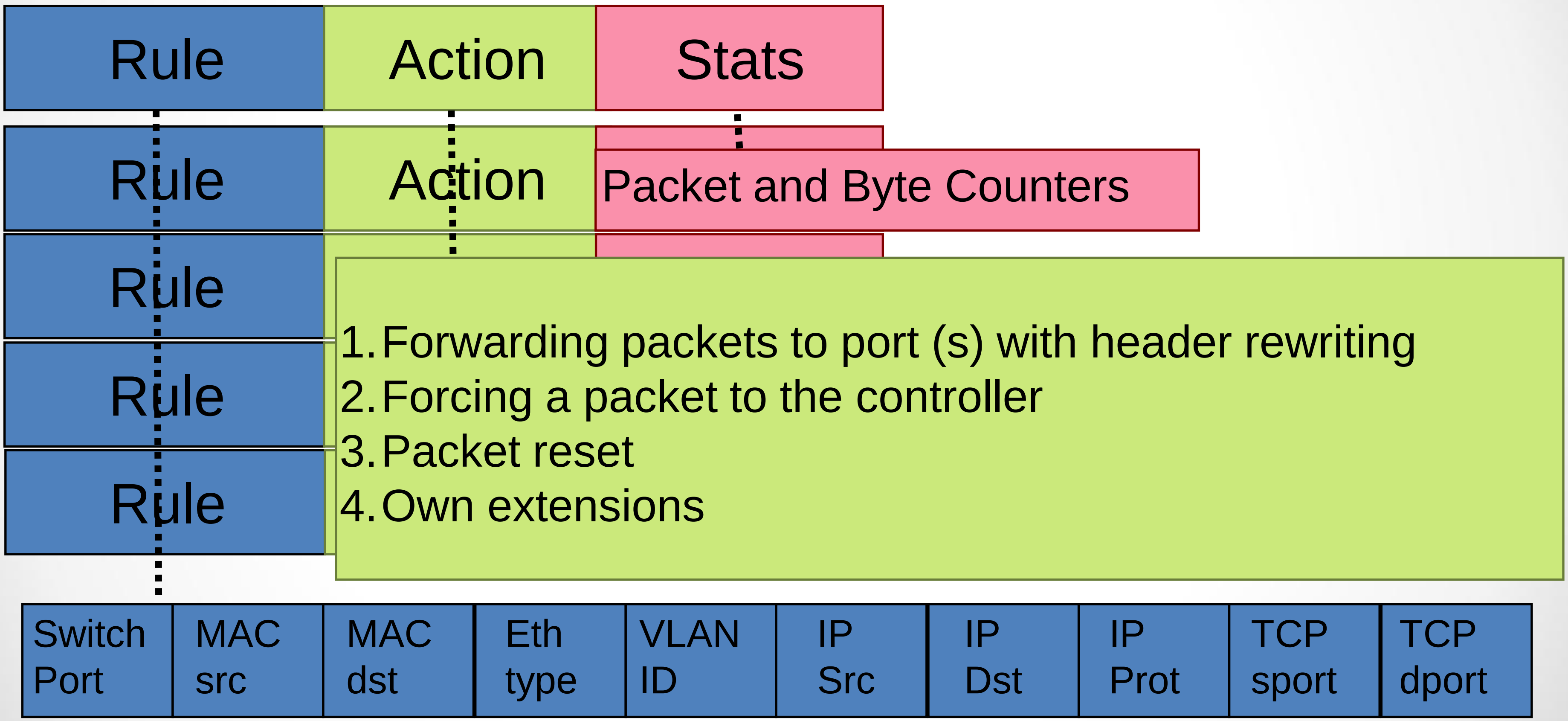
OpenFlow Protocol

Message Types:

- **Controller to Switch messages:**
  - Switch Configuration
  - Management and status monitoring
  - Stream Table Management
  - **Features, Configuration, Modify-State (flow-mod), Read-State (multipart request), Packet-out, Barrier, Role-Request**
- **Symmetric Messages**
  - Sending messages in both directions
  - Detecting Controller-Switch Connectivity Issues
  - **Hello, Echo**
- **Asymmetric Messages**
  - Sending messages from the switch to the controller
  - Announce a change in network status, switch status
  - **Packet-in, flow-removed, port-status, error**

OpenFlow 1.0

Flow Table



+ field mask

OpenFlow Rules

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

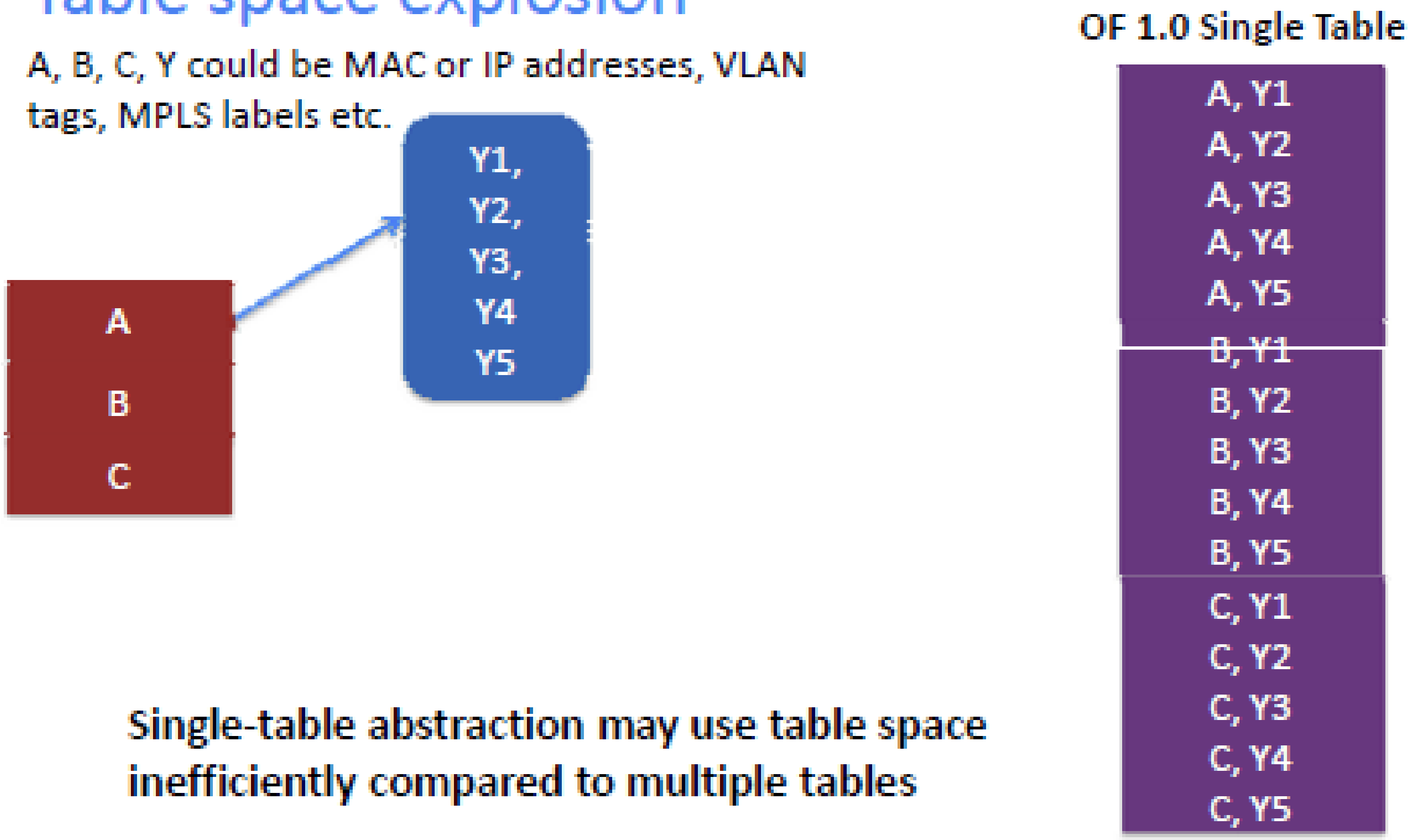
Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

One or multiple tables

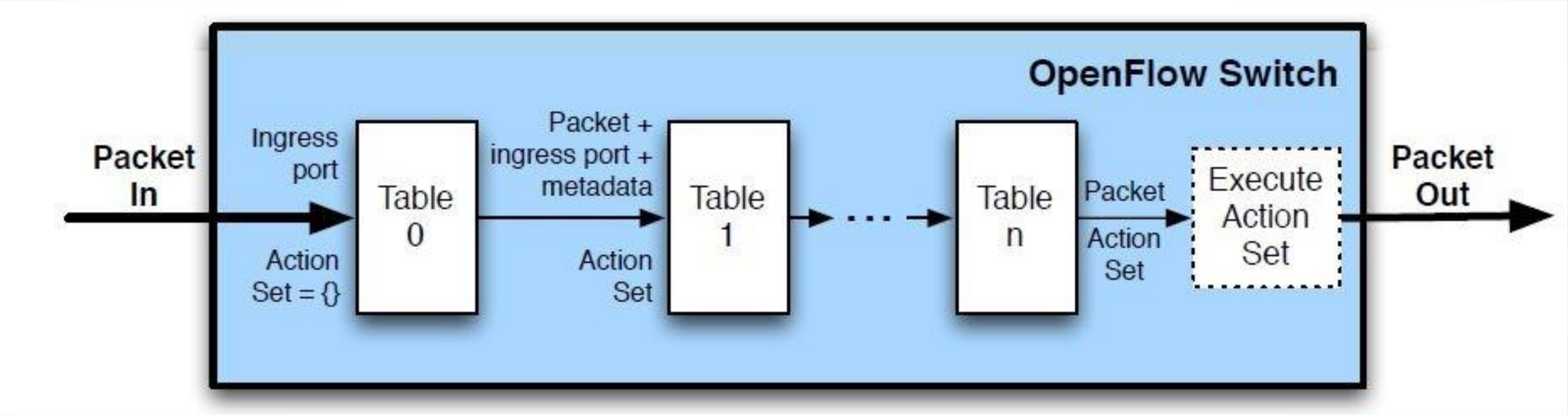
- Table space explosion

A, B, C, Y could be MAC or IP addresses, VLAN tags, MPLS labels etc.



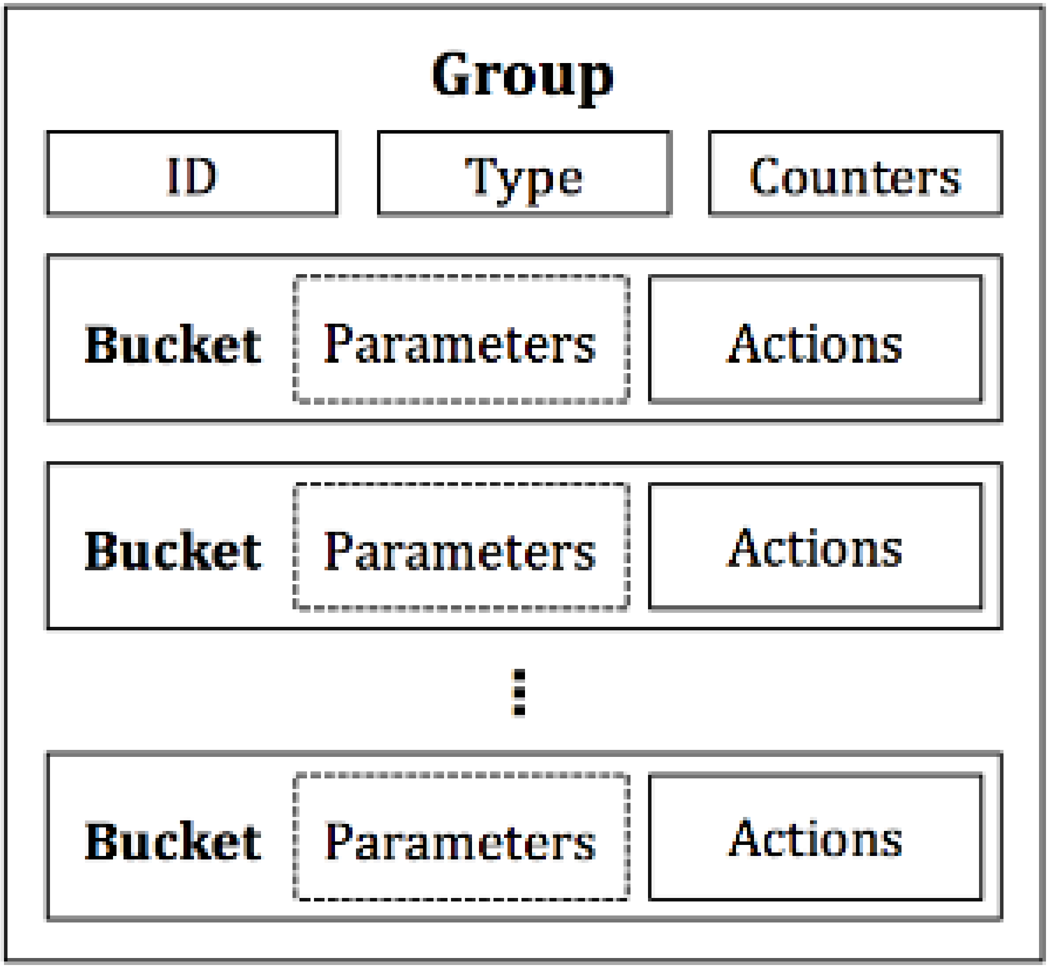
Single-table abstraction may use table space inefficiently compared to multiple tables

OpenFlow 1.1



- Moving packet only forward
- Transition: modifying a package, updating a set of actions, updating metadata

Group Table


Group
ID
Type
Counters
Bucket
Parameters
Actions
Bucket
Parameters
Actions
Bucket
Parameters
Actions

Meter Table

- To implement QoS and speed limits
  - For each thread or group of threads
  - Keeps track of counter values
  - Actions: **drop** or **dscp** remark

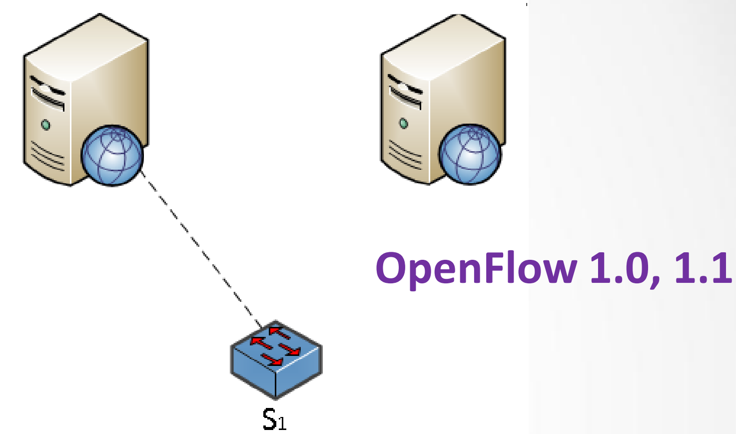
Meter Identifier	Meter Bands	Counters

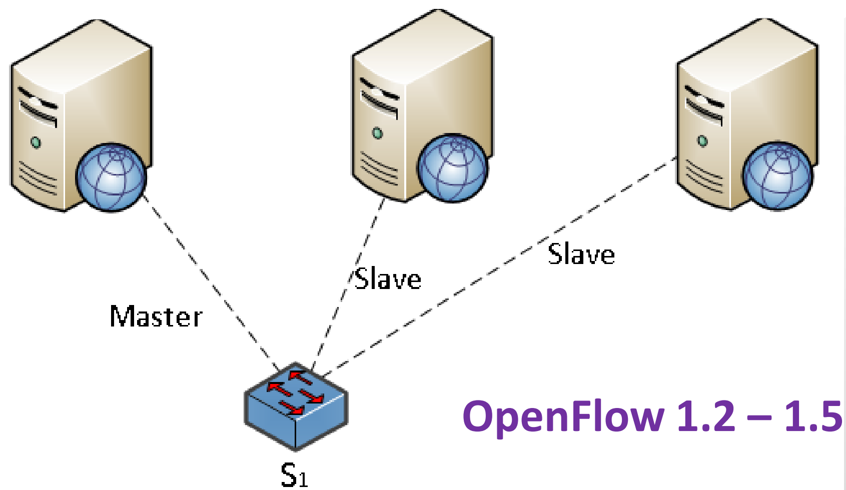
Band Type	Rate	Counters	Type specific arguments

Multiple Controllers

- OpenFlow 1.2:
  - Multiple controllers
  - Controller Roles

- ROLES:
  - **Roles:** Master, Slave, Equal
  - **By default:** each controller in Equal role for each switch.
  - **Role change:** OFPT_ROLE_REQUEST
  - **Role distribution**





OpenFlow Controller

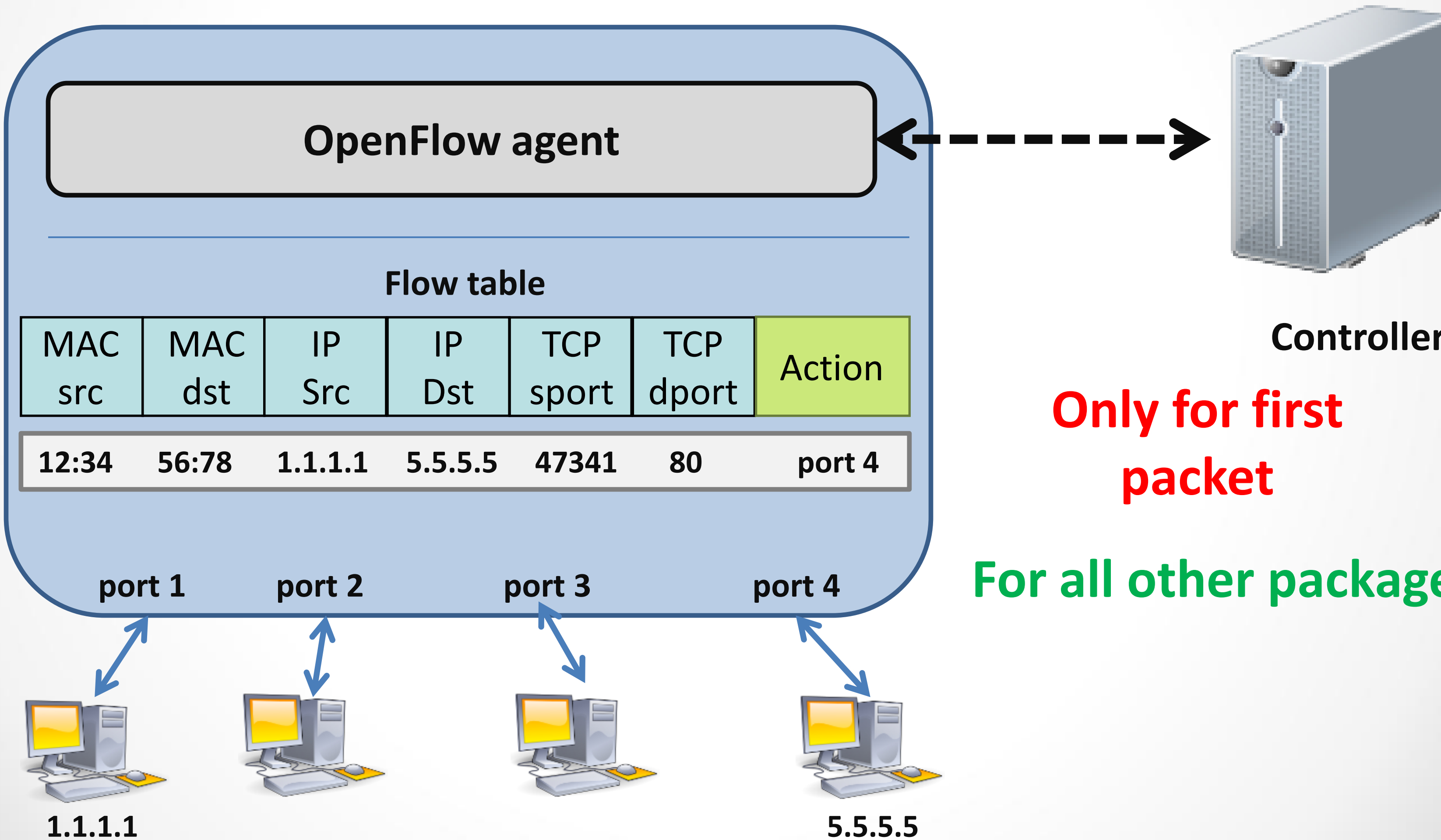
- Program, TCP / IP server, waiting for switch connections
- He is responsible for ensuring the interaction of the switch application.
- Provides important services (e.g. topology detection, host monitoring)
- The network OS API or controller provides the ability to create applications based on a **centralized programming model**.

OpenFlow Controllers List

- Controller Implementations:
  - Nox, Pox, MUL, Ruy, Beacon, OpenDaylight, Floodlight, Maestro, McNettle, Flower, Runos
  - Different programming form Python to Haskell, Erlang
- For educations and experiments - Pox.
- Developers Community
  - ONOS (Stanford)
  - OpenDayLight (Cisco)
- RUNOS Controller (ARCCN, Russia)
  - arccn.github.io/runos

Flow Installation Modes

Reactive mode



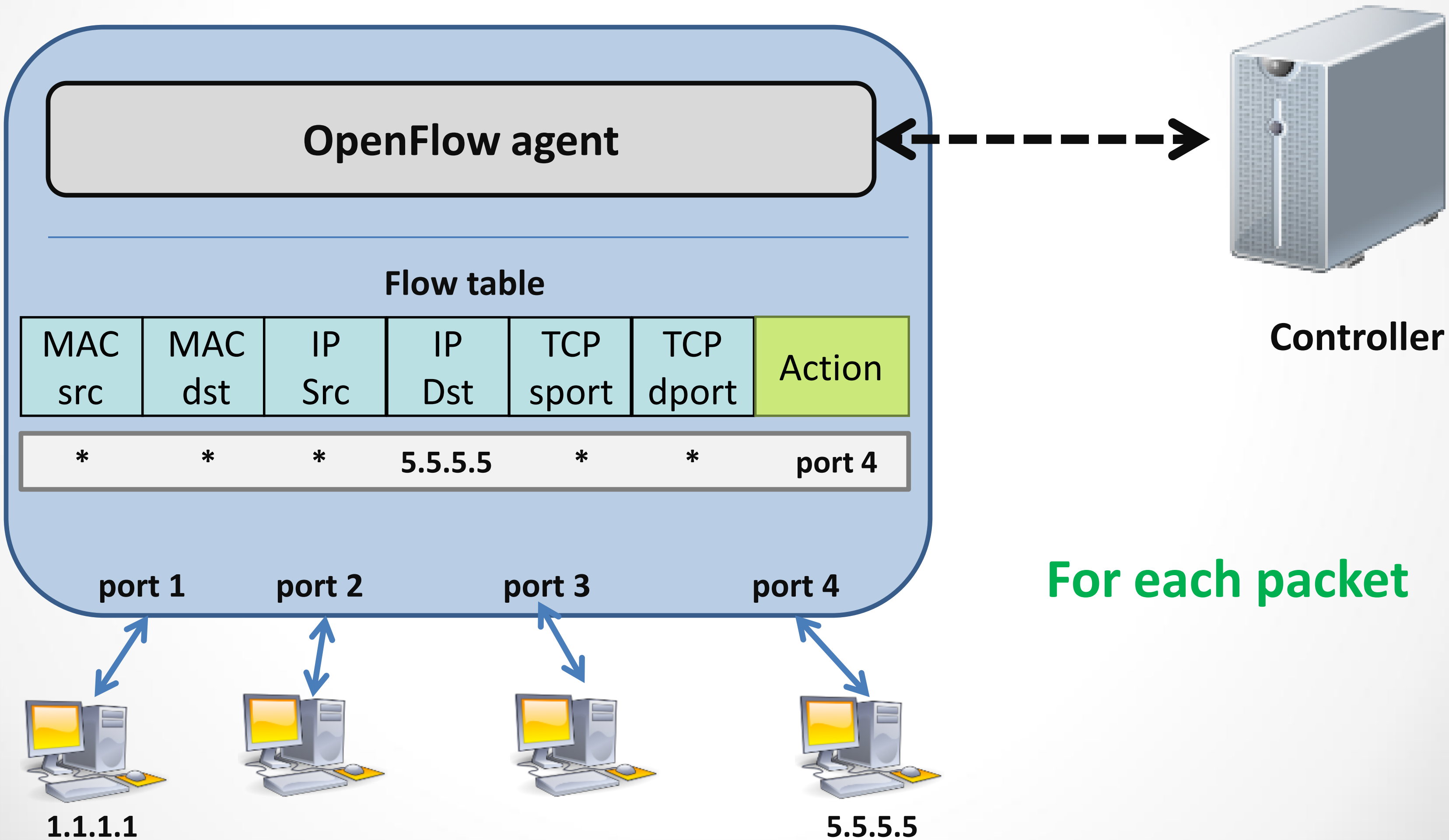
MAC src	MAC dst	IP Src	IP Dst	TCP sport	TCP dport	Action
12:34	56:78	1.1.1.1	5.5.5.5	47341	80	port 4

Only for first packet

For all other packages

Flow Installation Modes

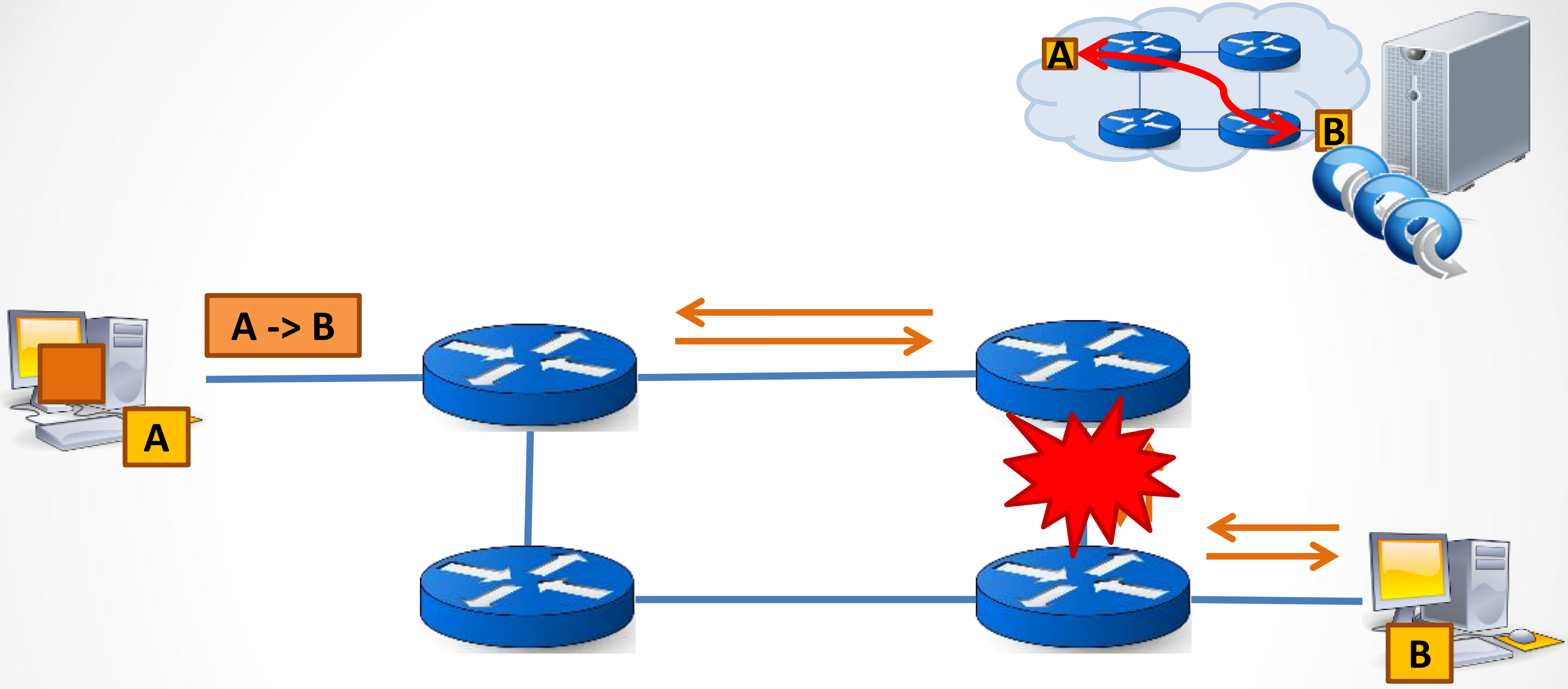
Proactive mode



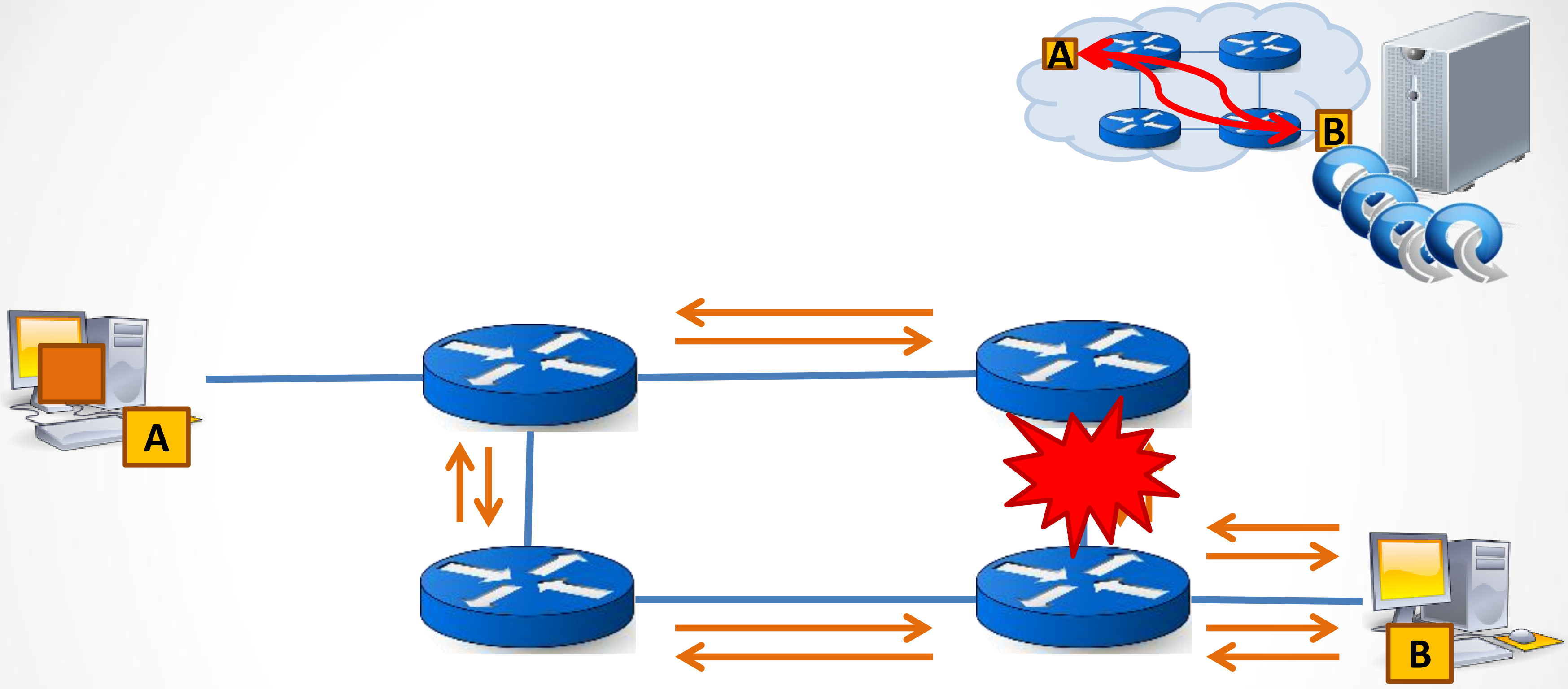
MAC src	MAC dst	IP Src	IP Dst	TCP sport	TCP dport	Action
*	*	*	5.5.5.5	*	*	port 4

For each packet

Routing in SDN/OpenFlow Network

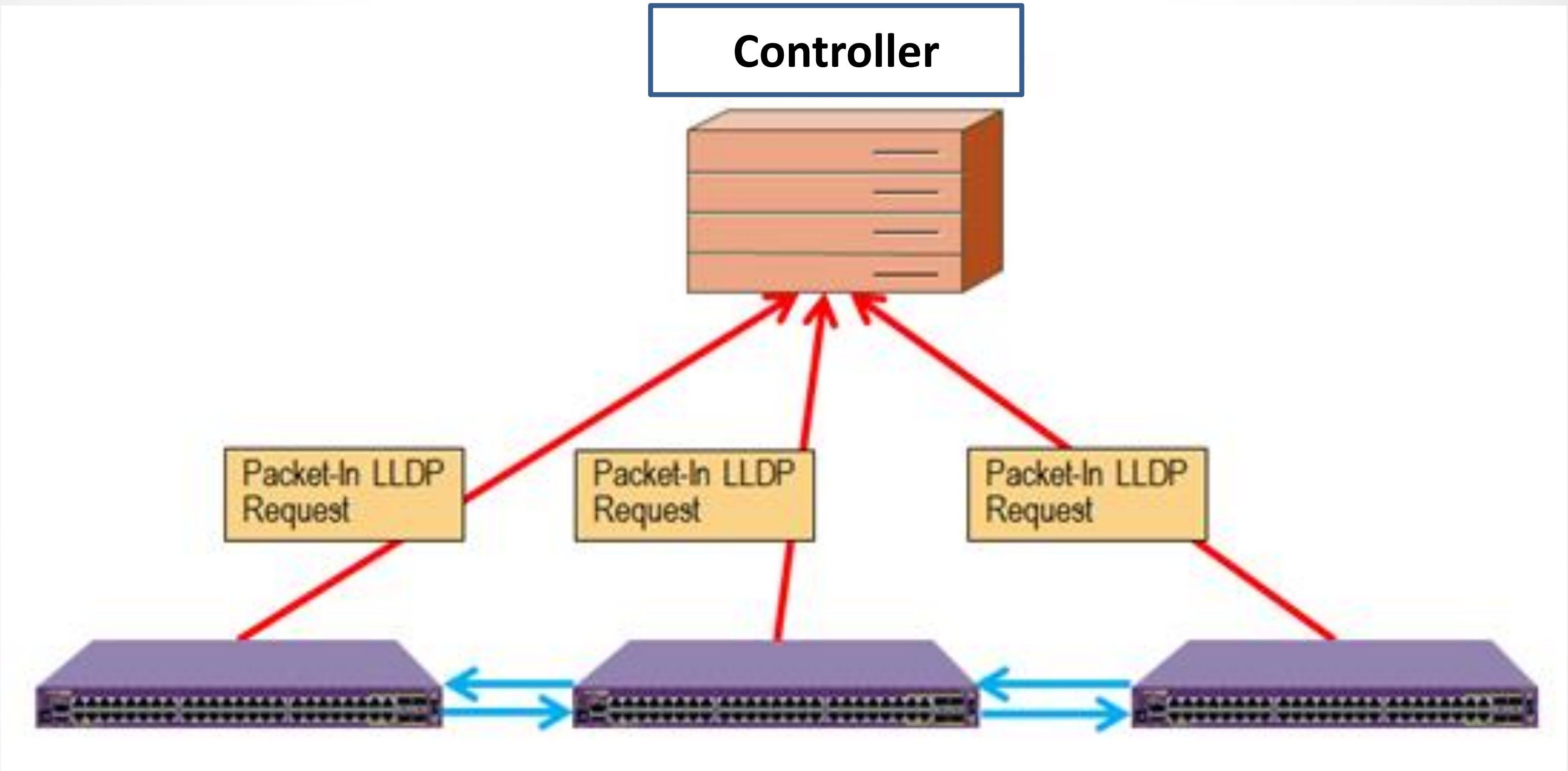


Routing in SDN/OpenFlow Network



Dynamic reconfiguration in the event of a network error.

Topology Detection in SDN/OpenFlow



OpenFlow Abstractions

- Control Plane – forwarding

- Data Plane – match-action tables

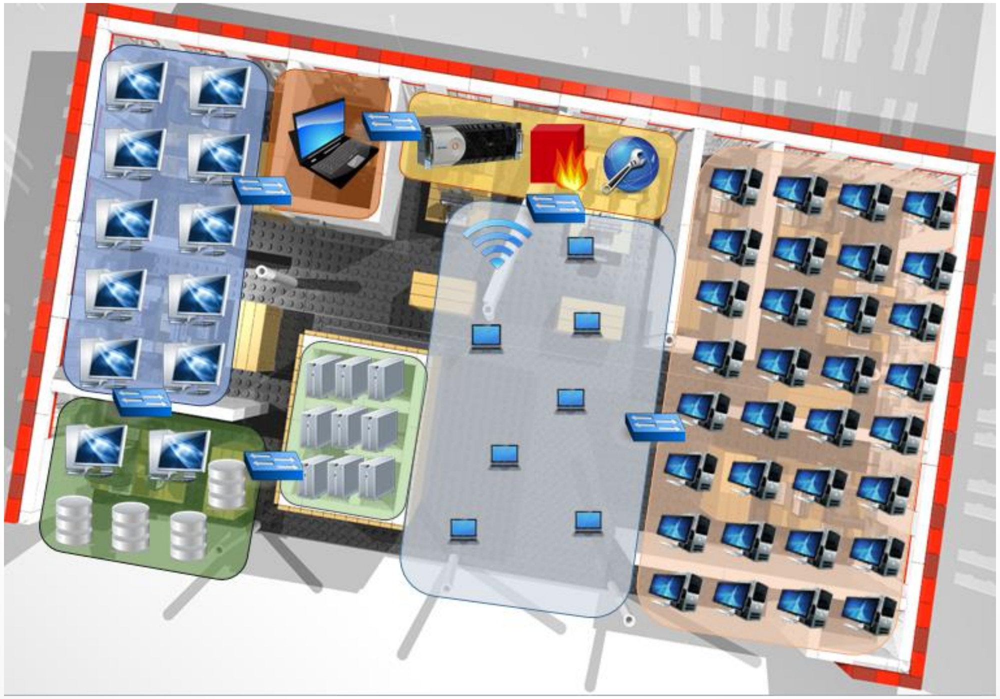
Part III: SDN/OpenFlow Use Cases

SDN Use Cases

- Companies
- Telecom operators and service providers
- Data center and clouds

Enterprise Network

- Modern companies have a complex network infrastructure:
  - A large number of network elements
  - Ramified topology
  - A set of different routing and security policies



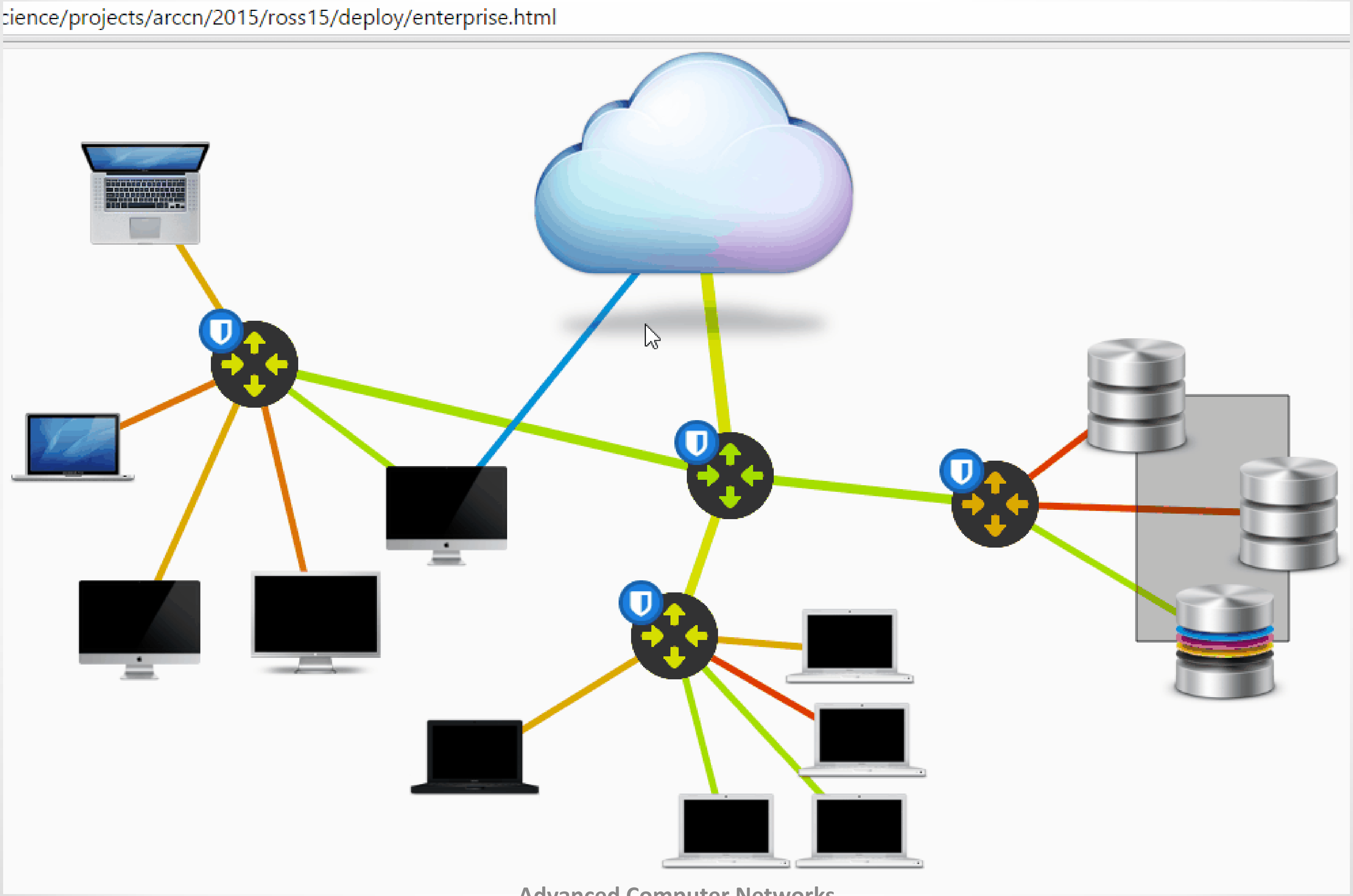
Administration difficulties

- Network administrators are responsible for maintaining the network infrastructure:
  - Network engineers translate high-level policies into low-level teams
  - Manual configuration of all network devices
  - Limited Network Device Management Toolkit
  - Retraining for each vendor
- There are so-called SNMP management systems, but only monitoring of the status, and not management - the configuration is still in manual mode.

Goals

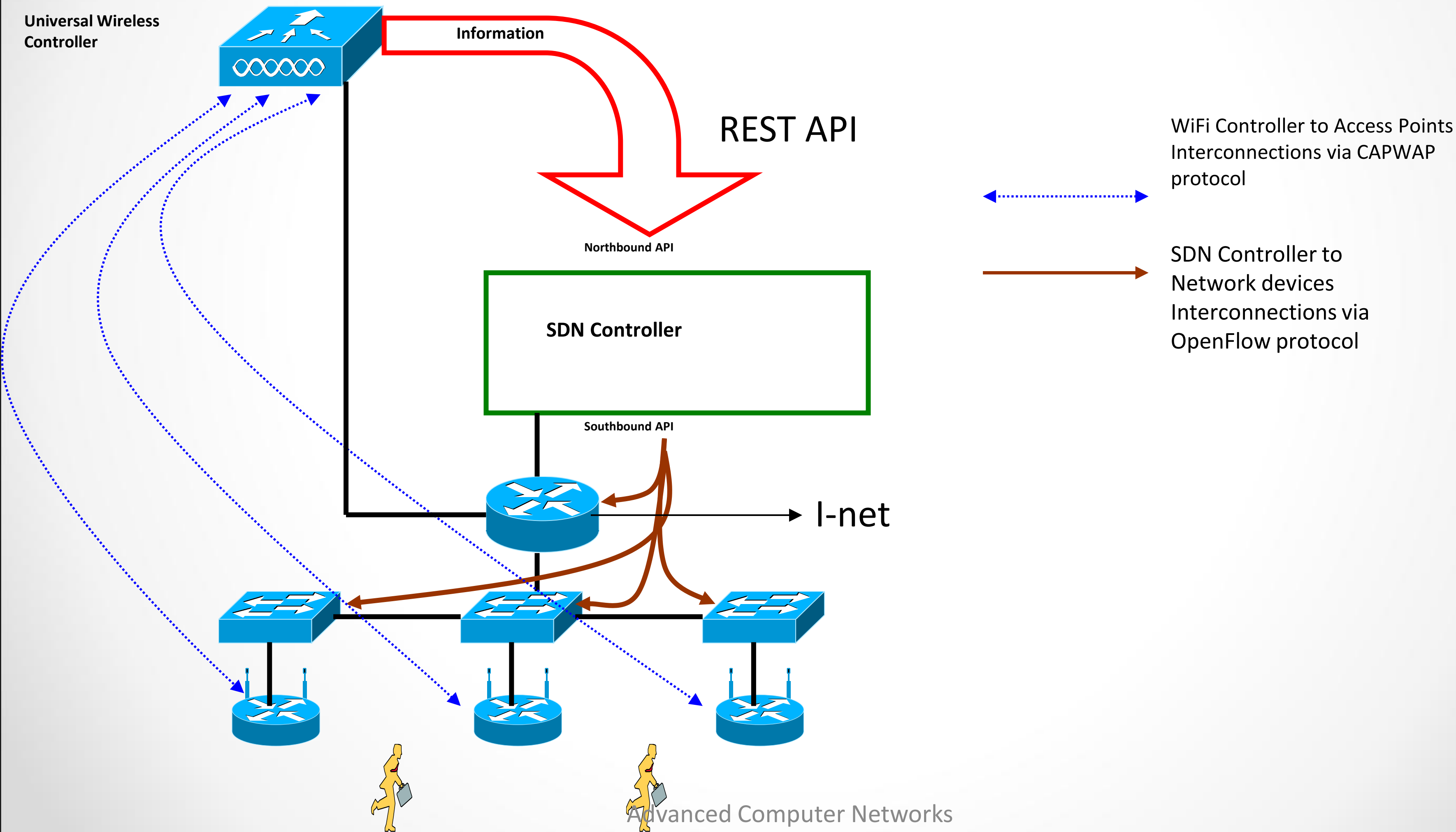
1. Make the network manageable without manual access to equipment.
2. Improve network management abstraction.

Web Interface



WiFi Controller & SDN Networks

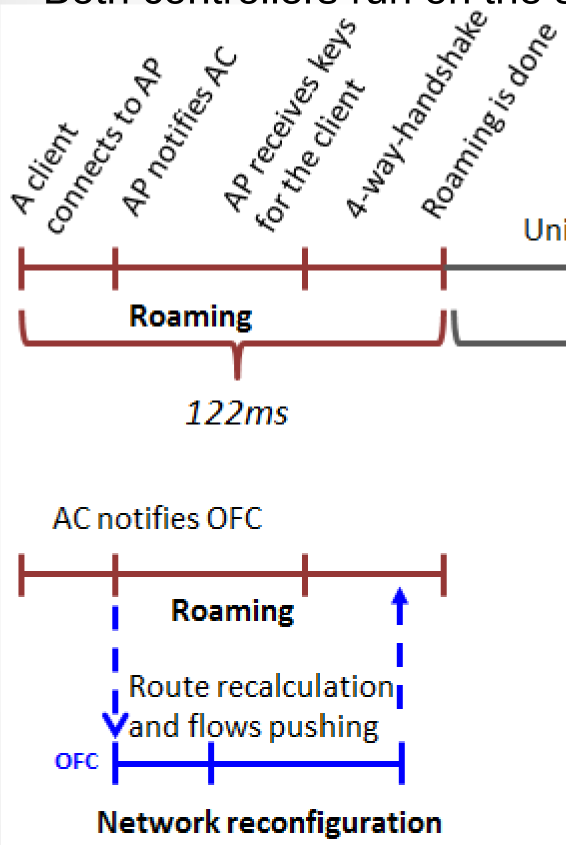
The principle of interaction with SDN controller over Northbound API:



WiFi Controller & SDN Networks

Test bed consists of:

- Three hardware OpenFlow switches Extreme Networks SummitX 460t
- Two TP-LINK access points
- A laptop moving from one AP to another one and running ping command to the outside server
- Both controllers run on the same server

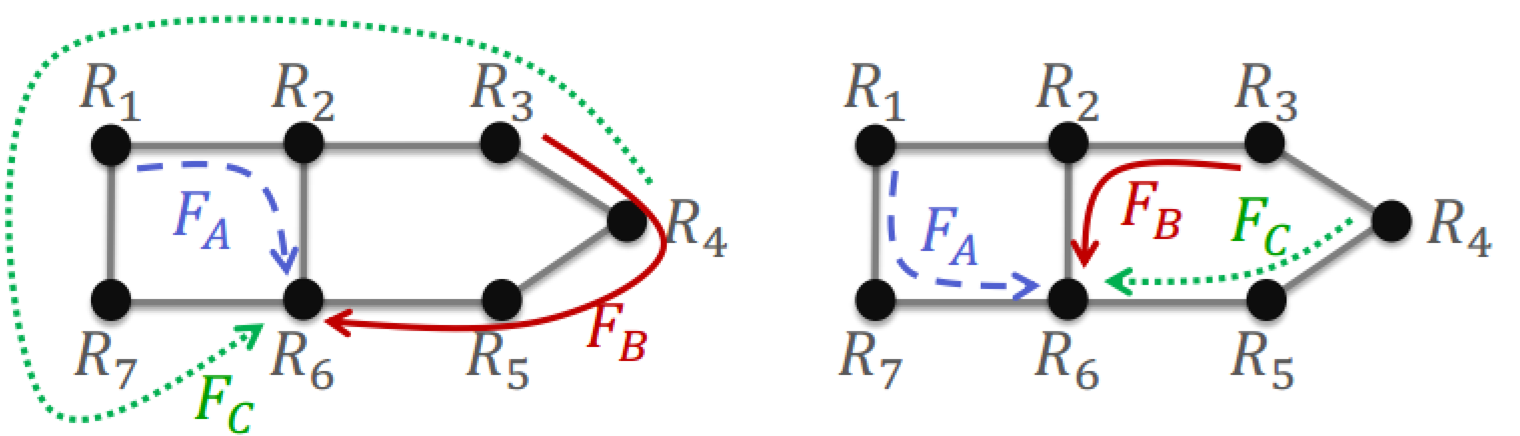


- The legacy network needs in average 1.5 seconds to reconfigure, while the SDN/OpenFlow network doesn't bring additional delay.
- This is because the migration procedure in Chandelle requires less than 80ms and the OpenFlow controller has enough time to reconfigure the switches.
- **Finally**, **we have more faster roaming with SDN/OpenFlow**.

Telecom

1. Intellectual Traffic Engineering:

- Choosing the best path in the graph
- Channel Failure Response
- Bandwidth Reservation

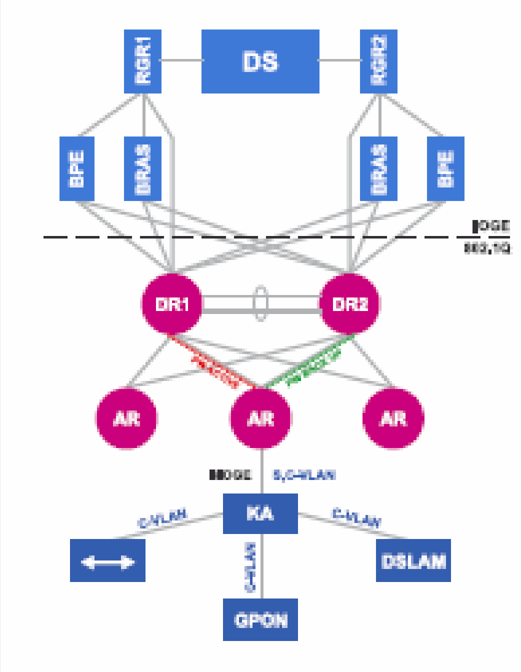


(a) Local path selection (b) Globally optimal paths

Telecom

- How to put all this into practice?
  - Greenfield?
  - Traditional Network Integration Issues
    - You need to play along with the protocols of the traditional network, i.e. answer queries correctly.
    - The fewer joints with a traditional network, the better.
  - The problem of integration with existing management systems

WAN segment (Service Provide)



Services:

- L2 transit for B2C, B2B/G (Internet, VPN)
- Fast backup path
- SLA
- IPTV multicast
- VoIP
- Mobile backhaul

Before:

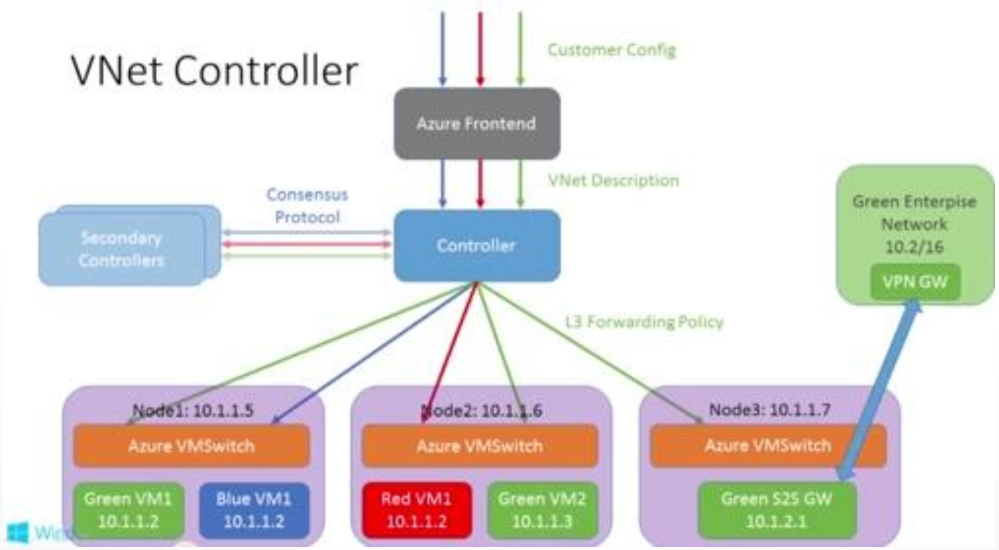
- IS-IS (RFC 1195)
- OSPF
- PIM-SSM
- PIM-SM
- LDP (RFC 3036)
- Targeted LDP
- BGP (PW для AToM)
- RSVP (RFC 2205)
- MPLS PW
- BGP (RFC 4271)
- MP-BGP (RFC 4760)
- MPLS-VPN (RFC 4364)
- MPLS (RFC 3031, 3032)

After:

- L2 PW application + stats (no encap)
- Bridge domain (no learning)
- Multicast (IGMP)
- L2 LAG, L3 ECMP
- H-QoS

Datacenters/Clouds

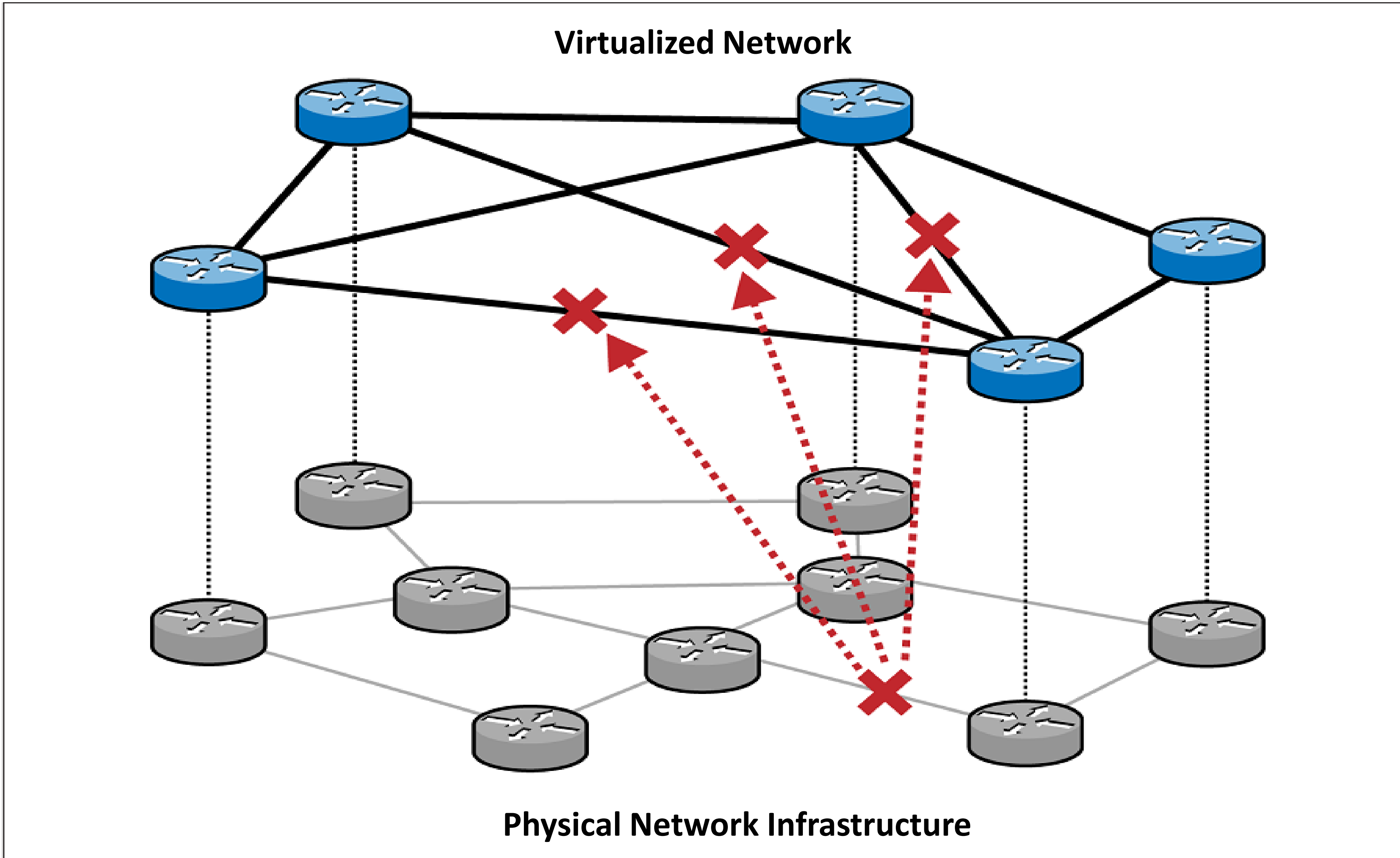
- Increased utilization of equipment and channels
- Monitoring and stream optimization
- User Network Virtualization
- Load balancing
- Quality Assurance



Datacenters/Clouds

- SDN:
  - Without OpenFlow, because they use OpenStack
    - ONLY virtual channels
    - Channels, tables, new VM, politics
  - With OpenFlow for managing of physical devices
    - Channel quality, bottleneck detection

Network Virtualization



Software Defined Data Center

Advantages

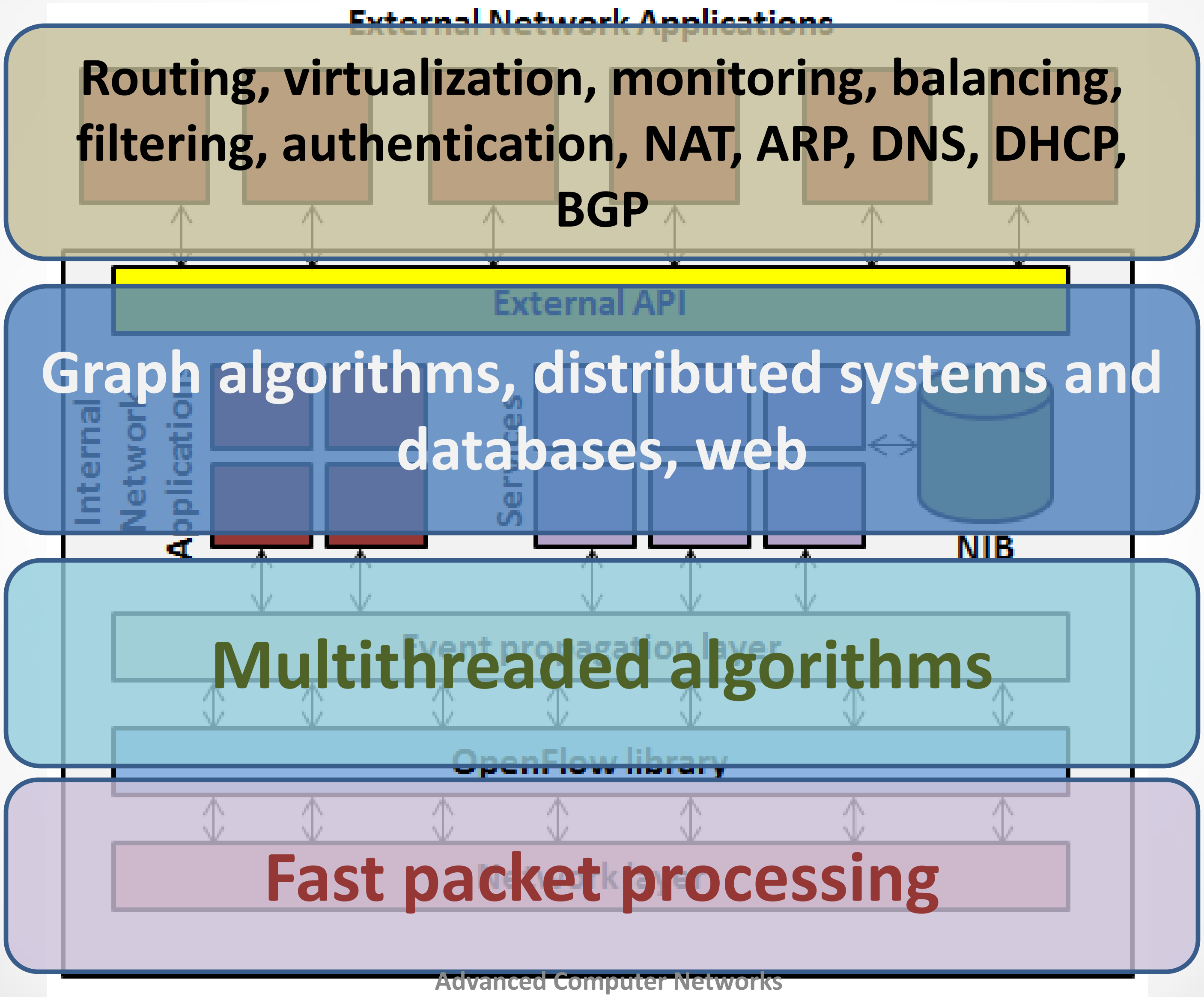
- Optimization of data center infrastructure management
- Reduced hardware dependency
- Automation of tasks
- Scalability, efficiency, flexibility
- Speed of implementation of the required functionality

Disadvantages

- The heterogeneity of solutions from different manufacturers
- Lack of mature standards
- The dampness of the proposed solutions
- Insufficient feature set

Part IV: OpenFlow Controllers

OpenFlow Controller Architecture



Requirements to SDN Controller

- **Performance**
  - Throughput
    - events per second
  - Delay
    - us
- **Reliability and security**
  - 24/7
- **Programmability**
  - Functionality: applications and services
  - Programming interface

List of OpenFlow Controllers(2013)

- NOX-Classic
- NOX
- Beacon
- Floodlight
- SNAC
- Ryu
- POX
- Maestro
- Trema

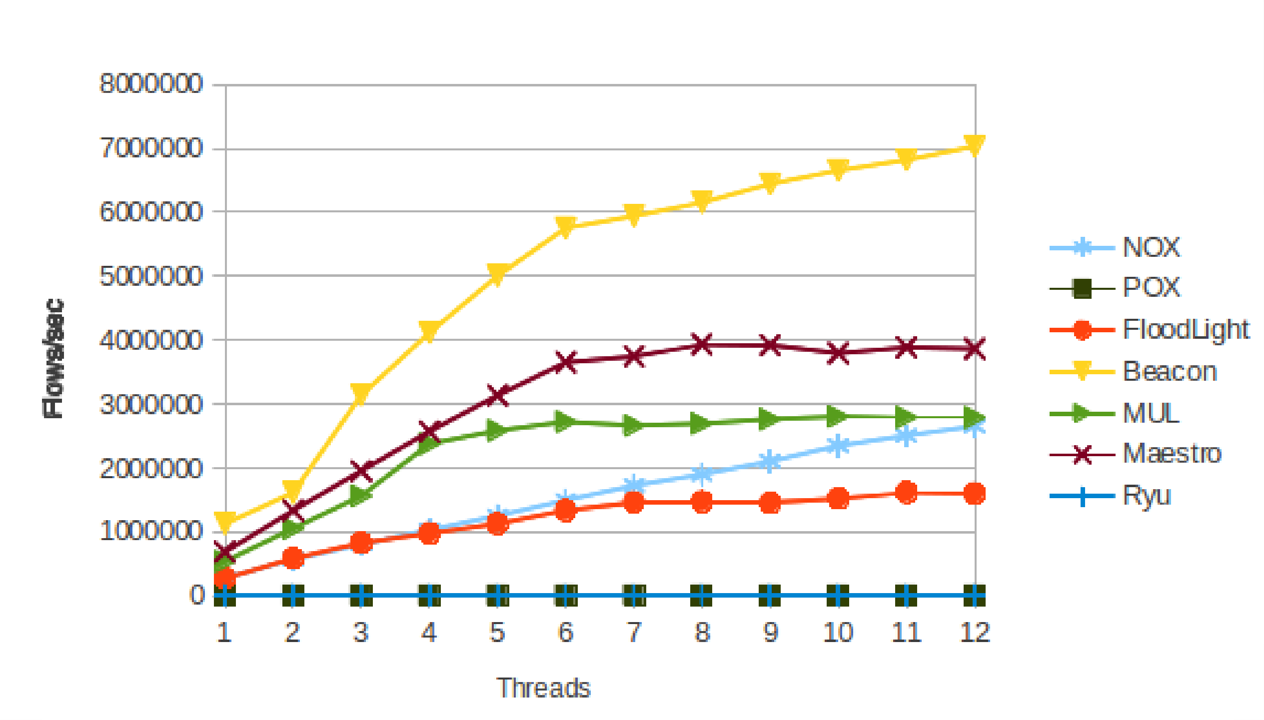
- Helios
- FlowER
- MUL
- McNettle
- NodeFlow
- Onix
- SOX
- Kandoo
- Jaxon

- Cisco ONE controller
- Nicira NVP Controller
- Big Network Controller
- IBM Programmable Network Controller
- HP SDN Controller
- NEC Programmable Flow

Experimental study

- Performance
  - maximum number of processing requests
  - request processing time for a given load
- Scalability
  - change in performance indicators with an increase in the number of connections with switches and with an increase in the number of processor cores
- Reliability
  - the number of failures during testing at a given load profile
- Security
  - resistance to incorrectly formed OpenFlow protocol messages

Comparison Results (2013)



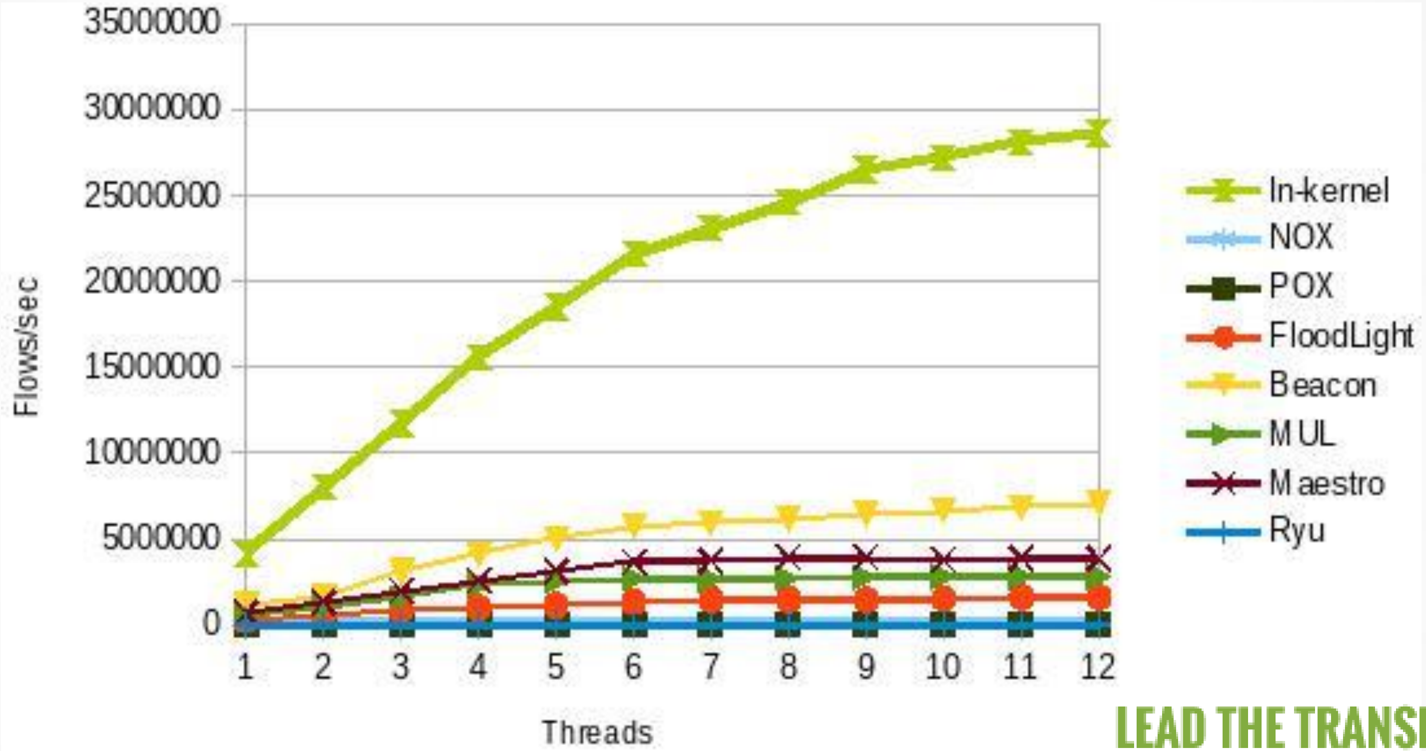
- Maximum throughput 7.000.000 flows per second.
- Minimum delay time from 50 to 75 µs.
- Disadvantages:
  - Reliability of controllers raised questions
  - Performance was not enough (DC> 10M fps)

Productivity increase

The most resource-intensive tasks:

- Interaction with OpenFlow switches:
  - multithreading usage;
  - thread loading accounting and rebalancing.
- Receiving OpenFlow packets from the channel:
  - reading packets from network card memory, bypassing the OS Linux network stack;
  - context switching;
  - virtual addresses.

Performance (In-kernel Controller)



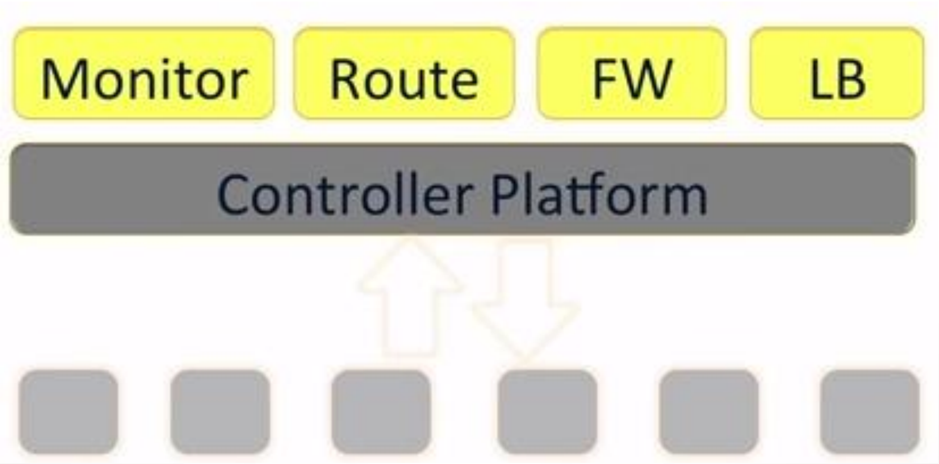
- Throughput: **30M fps**
- Delay: **45us**

Programmability

- In the controller language [fast]
- In any language through the REST interface [slow]
- Special programming languages with another abstraction (e.g. Pyretic, Maple)

NorthBound API

- NorthBound API - the interface between the controller and applications
- Programming with OpenFlow is not an easy task!
  - It is difficult to perform independent tasks (routing, access control)
  - Low level abstraction
  - You need to remember the rules on the switches.
  - Unknown rule order for switches
- Application Portability Between Controllers



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);
```

REST request example

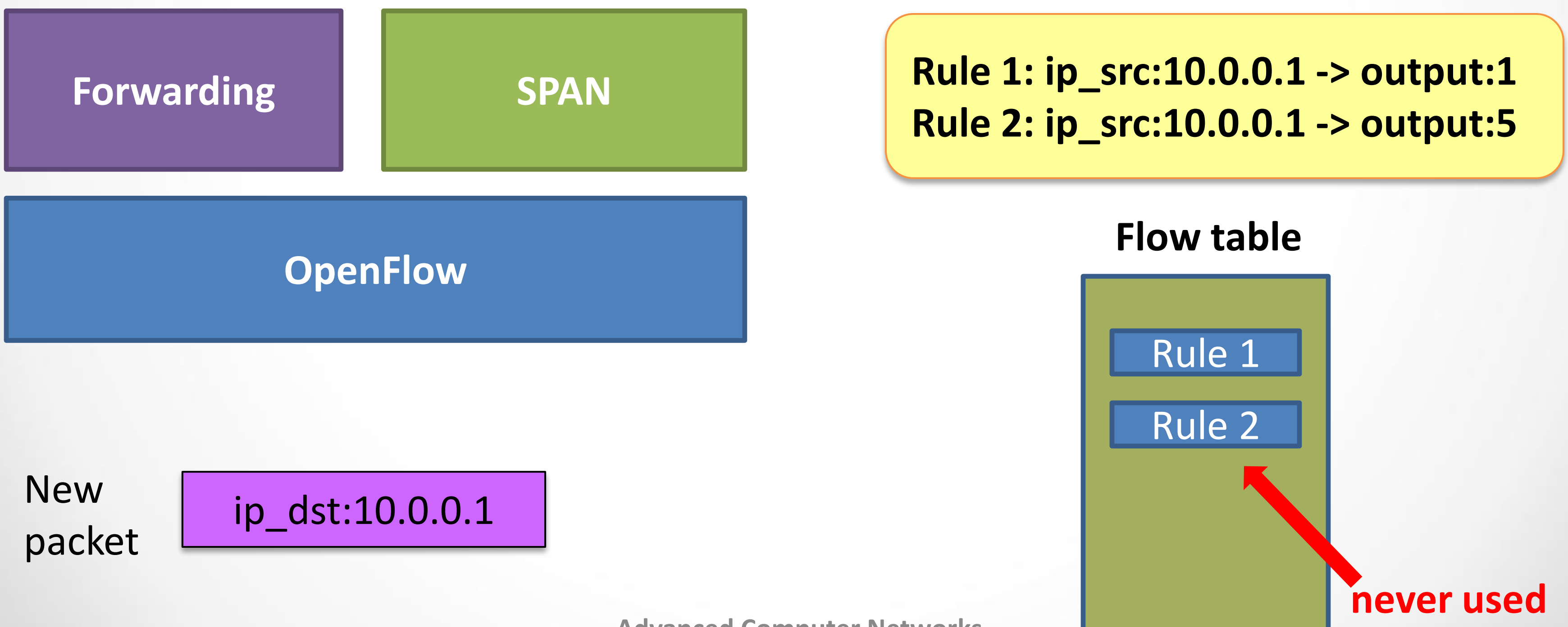
```
root@pc-1:~# curl http://127.0.0.1:8080/wm/staticflowentrypusher/list/00:00:00:24:e8:79:29:1a/json | json_pp -t dumper
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   670    0   670    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' => '45035997289868789',
                                                          'lengthU' => 72,
                                                          'length' => 72,
                                                          'outPort' => -1,
                                                          'xid' => 0,
                                                          'type' => 'FLOW_MOD',
                                                          'hardTimeout' => 0,
                                                          'idleTimeout' => 0,
                                                          'command' => 0
                                                        }
                                       }
        };
root@pc-1:~#
```

```
of13::FlowMod fm2; // Table 0: in_port,VLAN=ar.ep.stag -> output(ar.ep.port)
fm2.table_id(FORWARDING_TABLE);
fm2.priority(100);
fm2.cookie(cookie);
fm2.xid(mgr->impl->xid);
fm2.buffer_id(OFP_NO_BUFFER);
fm2.add_oxm_field(new of13::InPort(main_route[0].port));
fm2.add_oxm_field(new of13::VLANVid(end_switch_list[0].ep_list[0].stag | of13::OFPVID_PRESENT));
of13::ApplyActions acts2;
acts2.add_action(new of13::OutputAction(end_switch_list[0].ep_list[0].port, 0));
fm2.add_instruction(acts2);
mgr->get_connection(sw1_dpid)->send(fm2);

/* DR */
/* rules for the last one switch in the route */
of13::FlowMod fm3; // Table 0: VLAN=ar.ep.stag -> META=bbd_id, GOTO 1
fm3.table_id(FORWARDING_TABLE);
fm3.priority(100);
fm3.cookie(cookie);
fm3.xid(mgr->impl->xid);
fm3.buffer_id(OFP_NO_BUFFER);
fm3.add_oxm_field(new of13::VLANVid(end_switch_list[0].ep_list[0].stag | of13::OFPVID_PRESENT));
fm3.add_instruction(new of13::WriteMetadata(domain_id, 0xFFFF));
fm3.add_instruction(new of13::GoToTable(LEARNING_TABLE));
mgr->get_connection(sw2_dpid)->send(fm3);

// Table 1: META=bbd_id, in_port=dr.ep0.port -> TO_CONTROLLER
for (auto ep : end_switch_list[1].ep_list) {
    of13::FlowMod fm4; // Table 1: META=bbd_id, in_port=dr.ep0.port -> TO_CONTROLLER
    fm4.table_id(LEARNING_TABLE);
    fm4.priority(100);
    fm4.cookie(cookie);
    fm4.xid(mgr->impl->xid);
    fm4.buffer_id(OFP_NO_BUFFER);
    fm4.add_oxm_field(new of13::Metadata(domain_id, 0xFFFF));
    fm4.add_oxm_field(new of13::InPort(ep.port));
    of13::ApplyActions acts4;
```

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!**



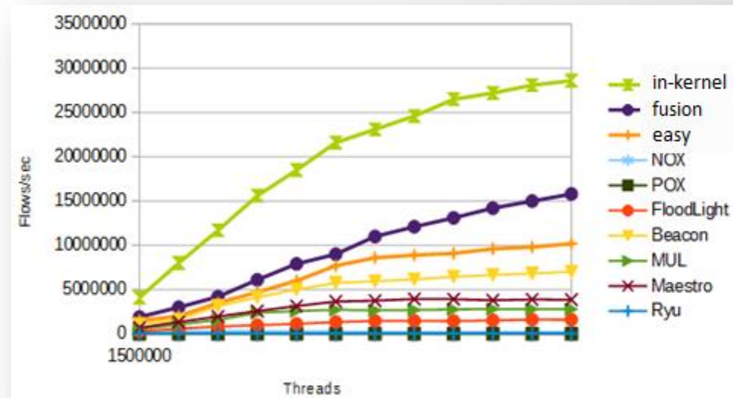
Part V: RUNOS Controller

RUNOS – Network Operating System

Network Management System - the first Russian SDN controller RUNOS

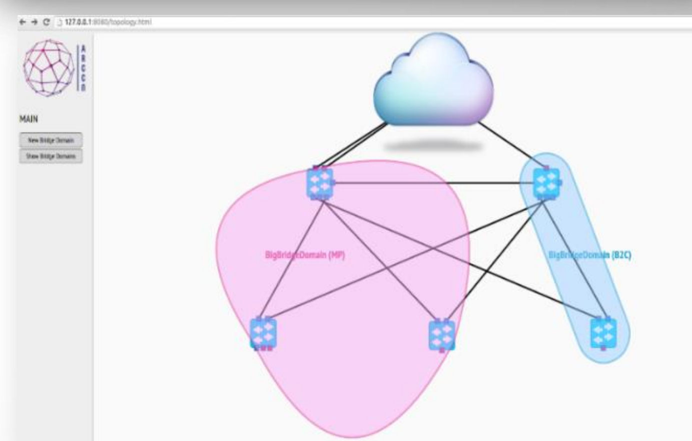
RUssian Network Operation System

There are different controller options with a single database and a different set of services and applications



RUNOS Open version

- on Github http://arccn.github.io/runos/
- Own base in C ++ 11/14, not Java
- goal: to simplify the development of network applications and not to forget about performance
- applications: topology, route, rebuilding in the event of a break, REST, WebUI, proactive loading of rules, redundancy Active-Passive



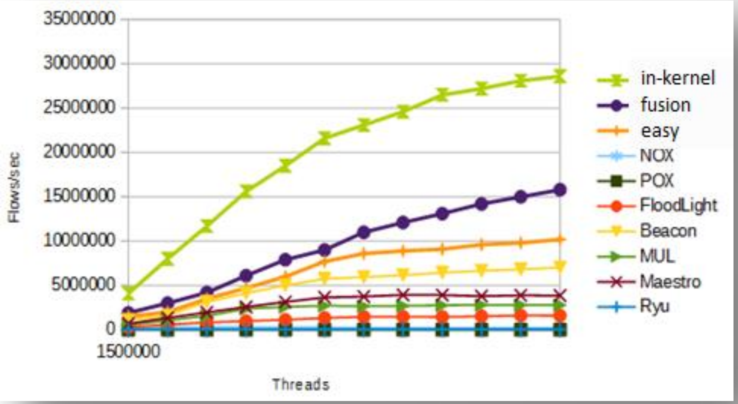
RUNOS – Network Operating System

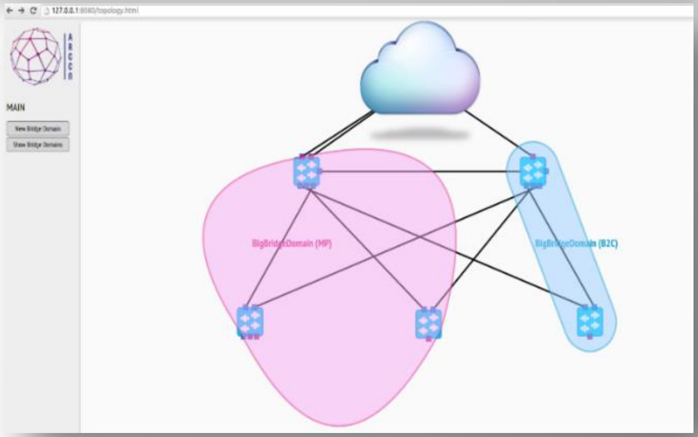
Network Management System - the first Russian SDN controller RUNOS

RUssian Network Operation System

There are different controller options with a single database and a different set of services and applications

- **In-Kernel RUNOS version**
  - Super performance 30 million events per second
  - Custom Application Development

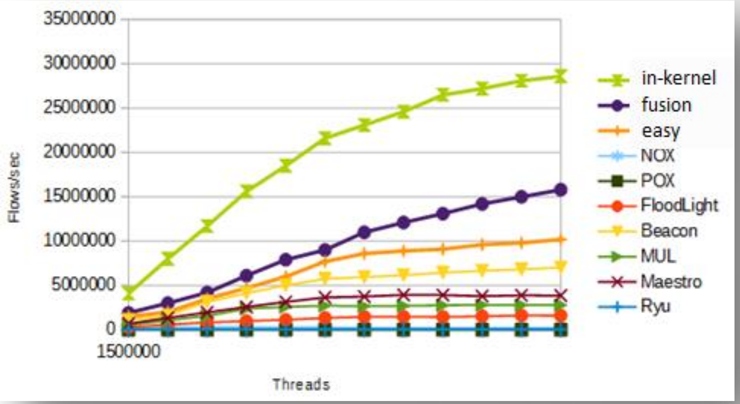




RUNOS – Network Operating System

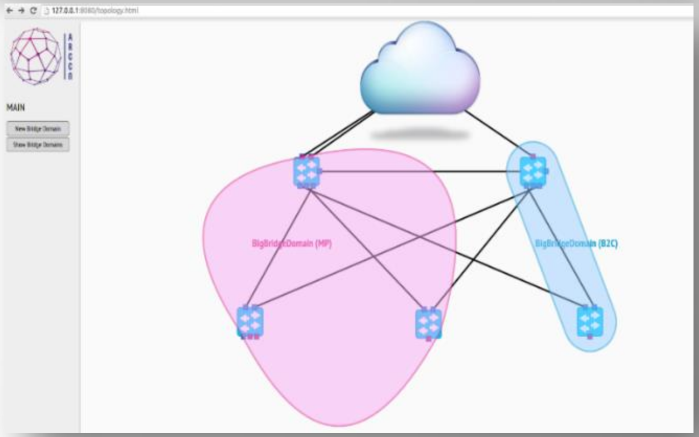
Network Management System - the first Russian SDN controller RUNOS

RUssian Network Operation System



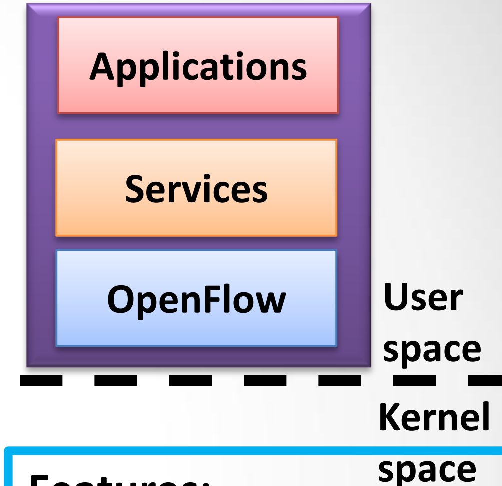
There are different controller options with a single database and a different set of services and applications

- **Commercial RUNOS version for telecommunications operators**
  - The base is the same as on Github. Customers can develop applications themselves. Learn from accessible materials
  - B2C, B2B services (p2p, mp2mp, multicast, etc.)
  - Active Standby Mode



RUNOS: Features

- The problem of launching multiple applications, integration with applications of other developers
  – Static tuning of applications for themselves is required, the order and method of transferring information between them.
  – There is no mechanism for controlling and resolving conflicts between applications (generation of overlapping rules).
- In RUNOS, the task is to solve the above problems:
  – part of the configuration occurs automatically according to meta information, linking occurs dynamically
  – conflict resolution system developed
  – A wide range of services to simplify the development of new applications

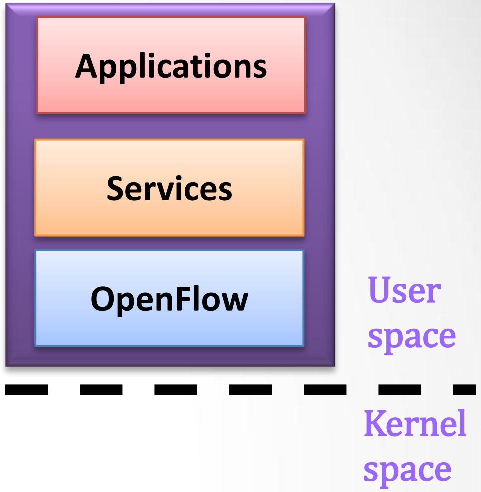


Features:
- Algorithmic policies (rule generation)
- Client-friendly API using EDSL grammar (low level details are hidden inside the runtime – overloading, templates)
- Modules composition (parallel and sequential composition)

Release Descriptions

Base:

- controller core
- topology building
- building a route through the entire network
- first version of the rule generation system
  - Priority allocation, combination of rules
  - LOAD, MATCH, READ abstraction
  - Based on maple
- Rest API (Floodlight compatible)
- WebUI (download monitoring, viewing tables, deleting and adding rules)
- Proactive loading rules
- Cold reservation
- ARP caching



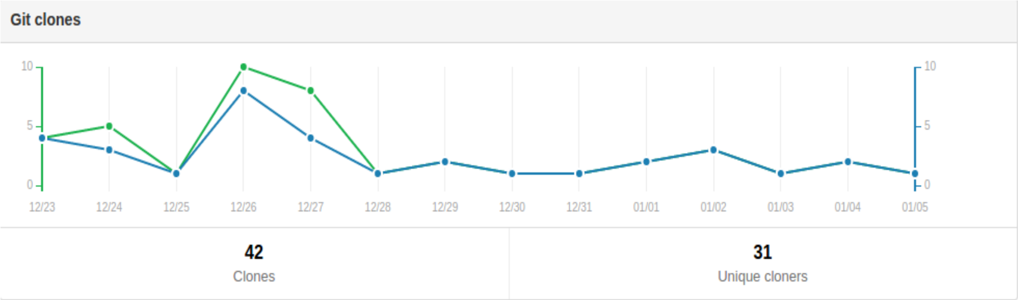
Release Descriptions

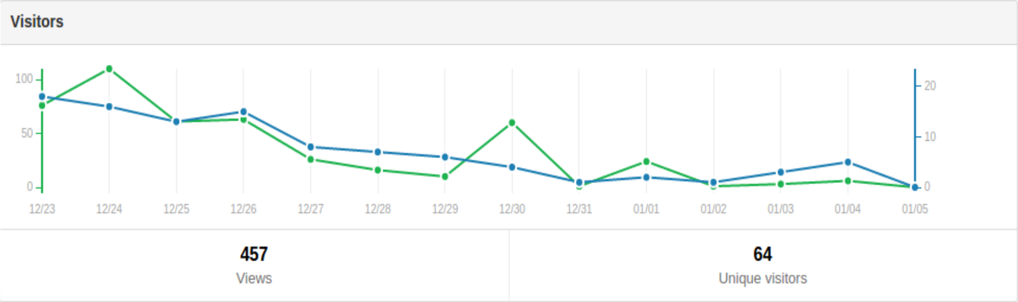
Version 0.6 is one of the last big releases.

- Full update of the controller core structure. There is no binding to a specific version of the OpenFlow protocol. Own model, expandable for any new fields, including those specific to equipment.
- Batch grammar for network applications. Simplifies the development of new applications.
  - "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()"
- Updating the rule generation system - increased speed and improved rule generation (by the number of rules and the number of priorities).
- Test system.
- Runos-book detailed documentation and instructions for developing new applications.
- Applications: stp, arp, flow-manager

Open source RUNOS

- Sources: http://arccn.github.io/runos/
  - Apache, version 2.0
- Tutorial s(Readme.md + Runos-book)
  - building, installation, running
  - First application tutorial
- Virtual Machine

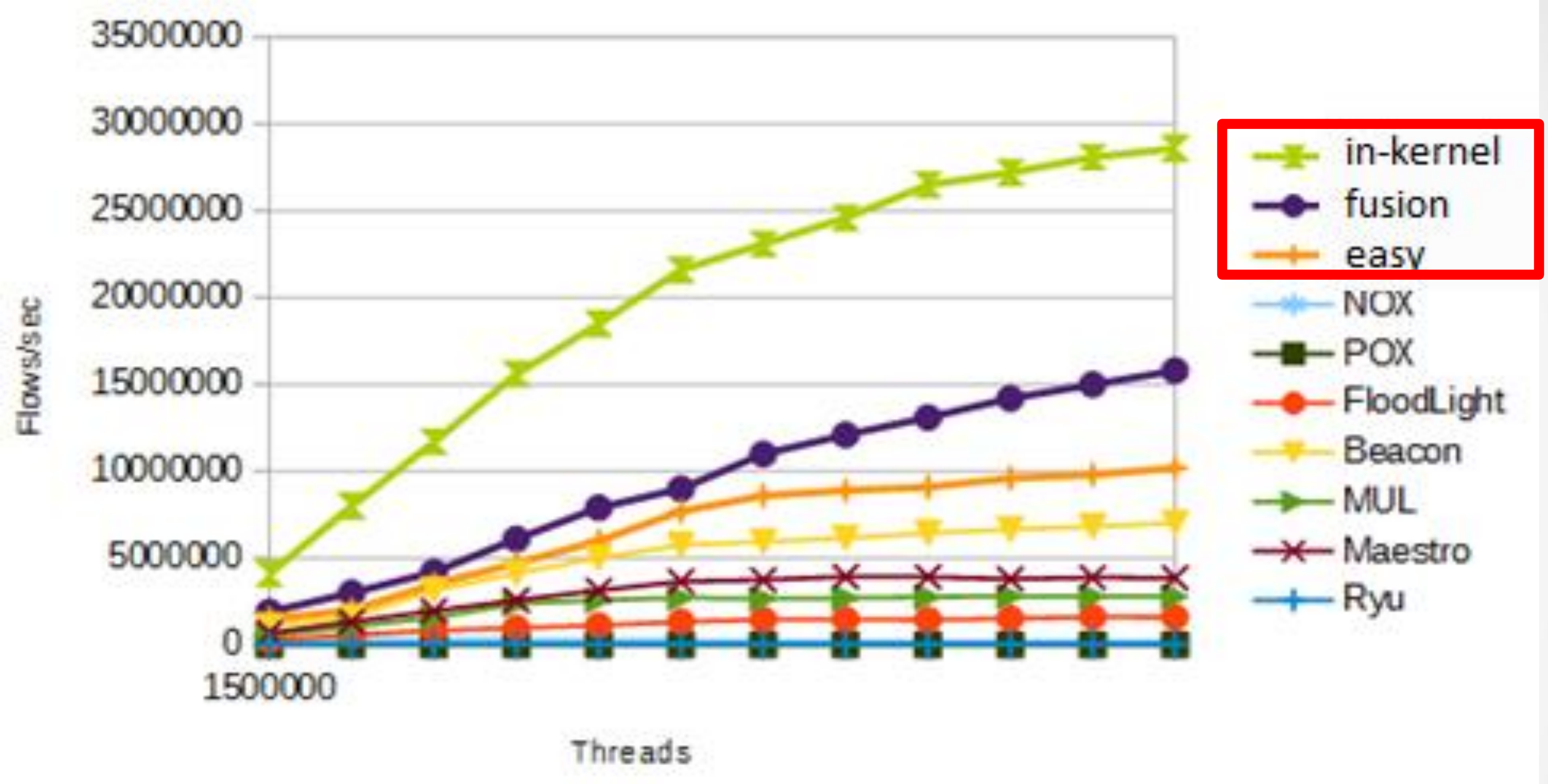




New RUNOS Releases

- v0.6.1
  - Clean OpenFlow interface for programming switches (the ability to create rules yourself)
  - Updated REST: compatibility with Ryu, a library for Postman
- v0.7
  - Optimization of the rule generation system:
    - Global network vision
    - Optimization of work - by number of FlowMod
    - New applications: corporate network
  - Improving the Web interface (transferring part of the functionality from the commercial version)

Performance



- Throughput: **10 000 000 flows per sec**
- Delay: **55 µs**

LEAD THE TRANSFORMATION
OPEN NETWORKING SUMMIT 2014
MARCH 3 - 5, 2014
SANTA CLARA CONVENTION CENTER & HYATT

Implementation

Keywords: C ++ 11/14/17, QT, Boost (asio, proto, graph)

The main third-party components:

- **libfluid project** (_base, _msg)
  - for interaction with switches and analysis of OpenFlow 1.3 messages
- **libtins**
  - parsing packets inside OpenFlow messages
- **glog (google log)**
  - multithreaded logging
- **tcmalloc (google performance tools)**
  - alternative faster implementation of malloc / free
- **json11**
  - parsing the configuration file
- **boost graph**
  - Topology storage, route search

Parameters

Config (json):

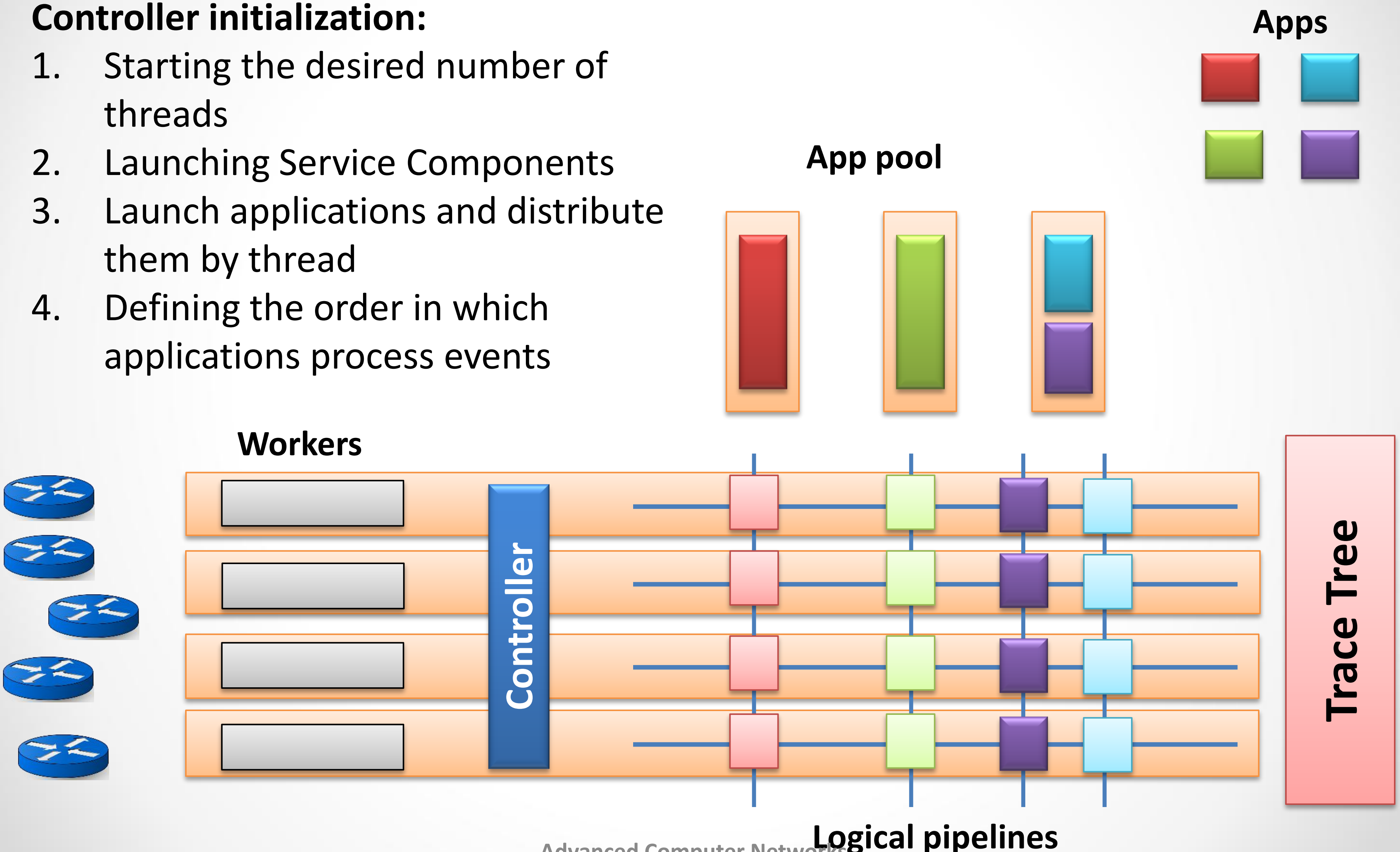
```
"controller": {
  "threads": 4
},
"loader": {
  "threads": 3
},
"link discovery": {
   "poll-interval" : 10,
   "pin-to-thread" : 2
},
"learning switch": {
}
...
```

- **The number of controller threads is set**
  - for interacting with switches
  - for applications
- **Application list**
  - their parameters (poll-interval)
  - lock the thread of execution or select for exclusive use (pin-to-thread, own-thread)

Architecture

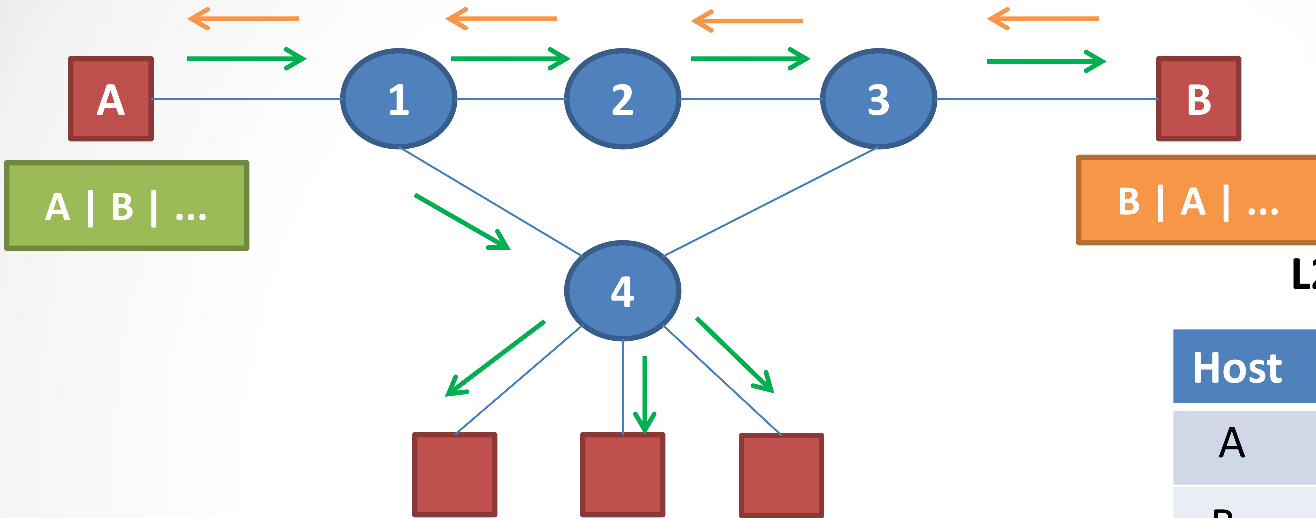
Controller initialization:

1. Starting the desired number of threads
2. Launching Service Components
3. Launch applications and distribute them by thread
4. Defining the order in which applications process events



Part VI: Application Development for RUNOS Controller

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;
}
```

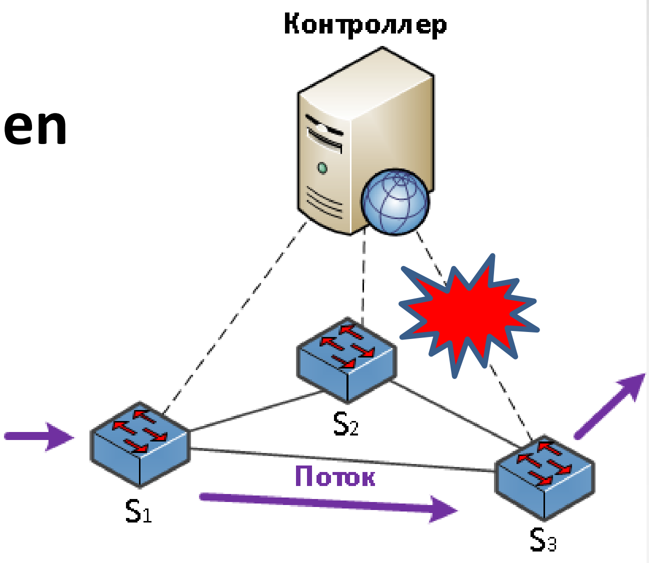
Part VII: Distributed Control Plane

High Availability (HA)

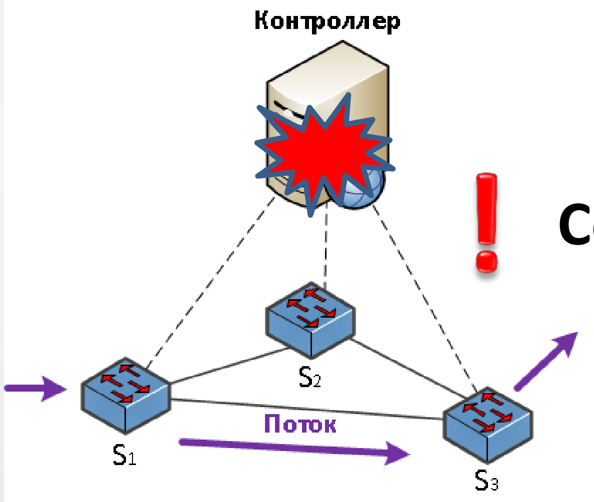
- The network operates in 365/24/7 mode.
- The SDN control platform must be running continuously.
- The goal of high availability is to support the continued availability of the management platform and network applications.
- Causes of downtime: maintenance, software and hardware errors, hardware failure, attack, power outage, accident.

Availability koefficient, %	Downtime
99,999	5 min
99,99	52 min
99,9	8,7 h
99	3,7 days

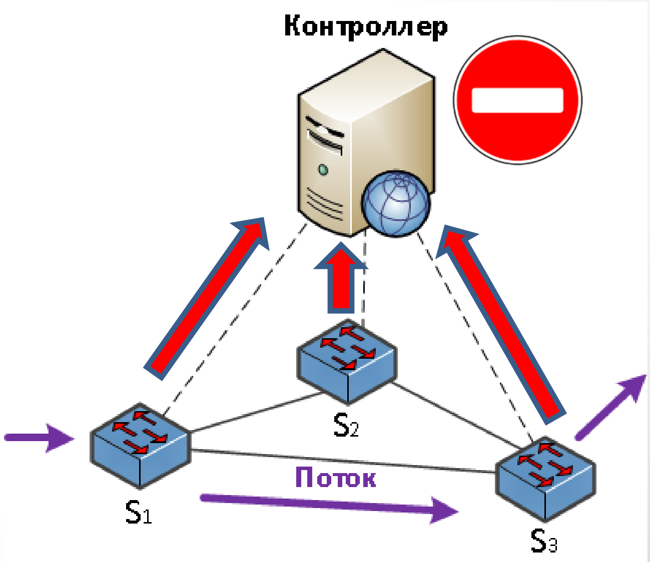
Failures



! **Loss of connection between switch and controller**

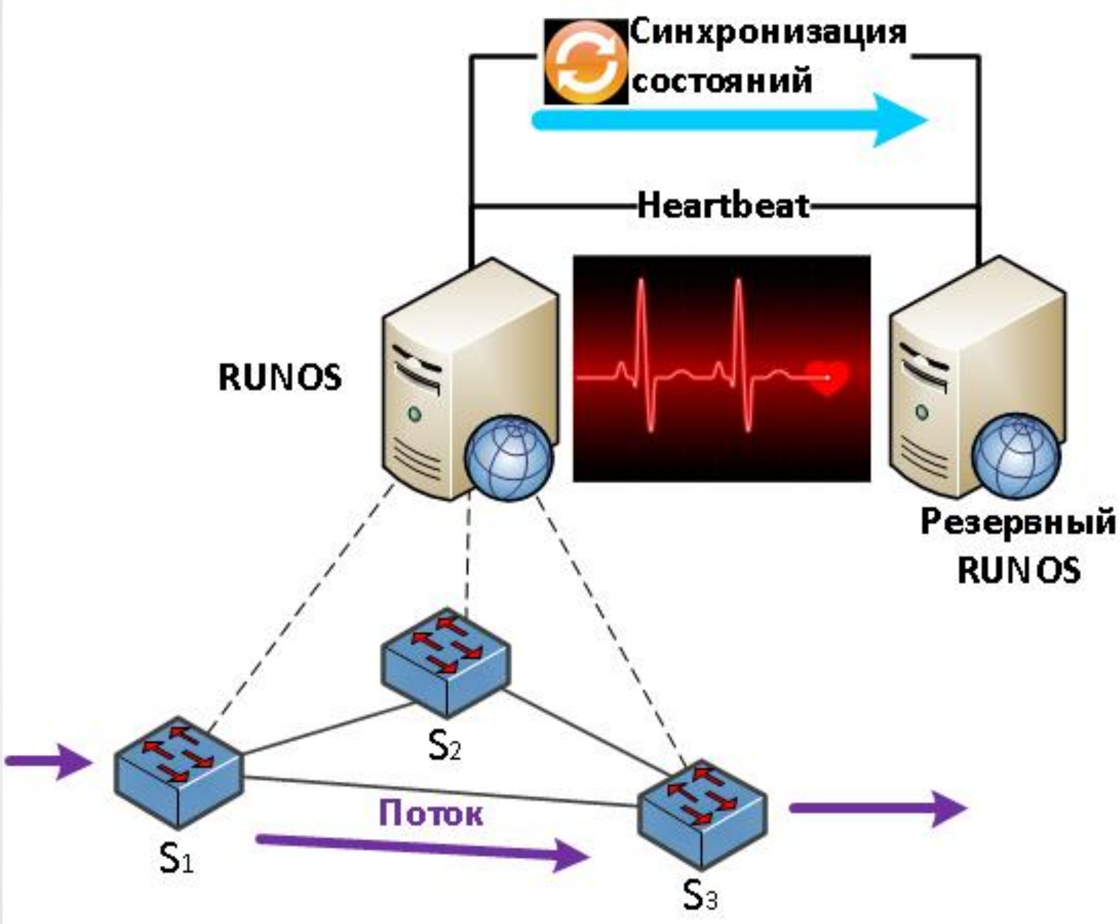


! **Controller Failure**



! **Controller Overloading**

HA Active/Standby Techniques /1



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

- **Cold**

[no sync]

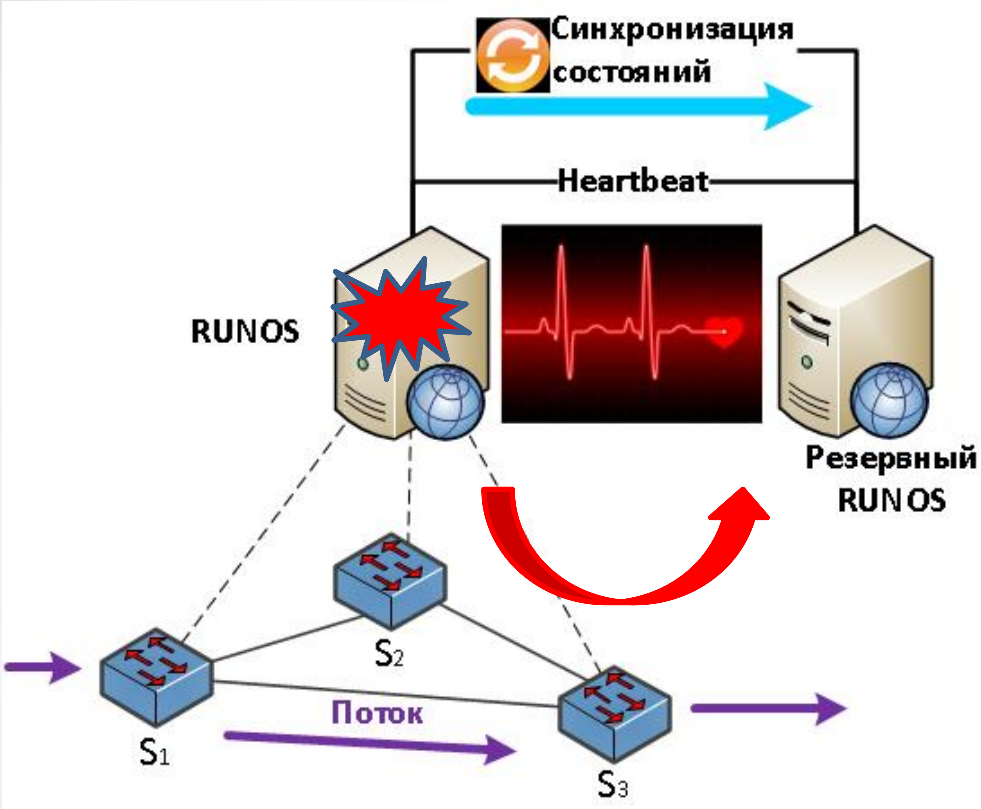
- **Warm**

[periodic synchronization]

- **Hot**

[continuous synchronization]

Restoration

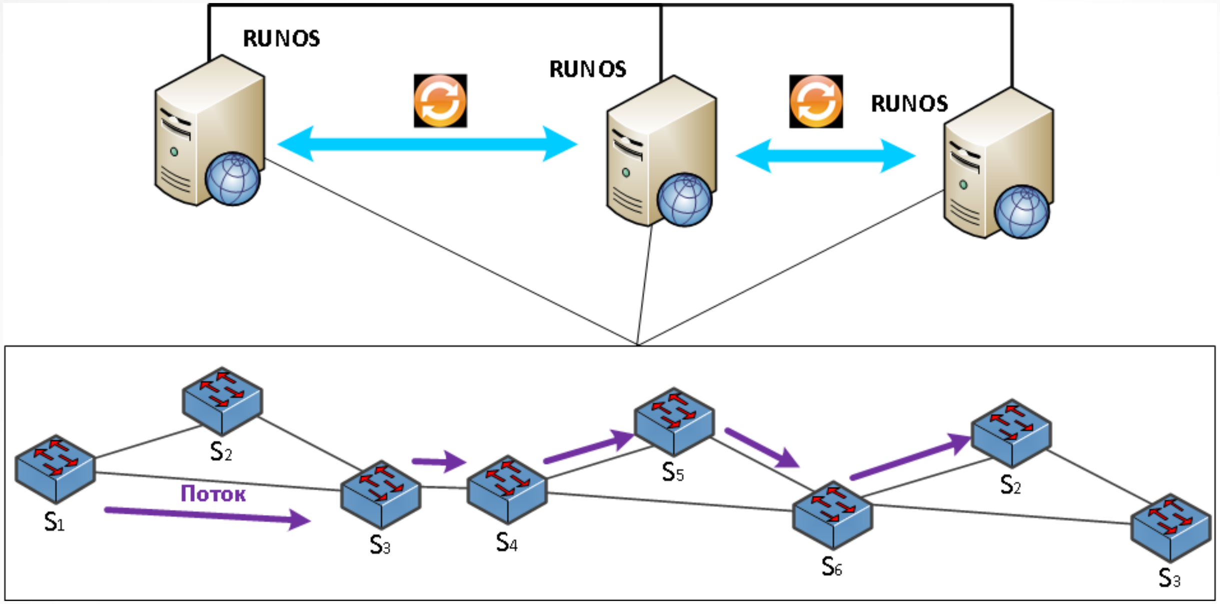


Features of the solution:

- OpenFlow ≥ 1.0
- Corporate networks.
- Does not scale
- Incomplete utilization of computing resources

+ Single controller failure
- Loss of connection between switch and controller
- Controller overload

HA Active/Active Techniques /2



- Active / Active Backup Strategies
- Asymmetric
- **Symmetrical**
- High complexity [Requires coordination of controllers, global state support]
- High Availability [Minimum Downtime]
- High utilization of computing resources

Conclusion

- SDN is already actively used in the industry and is the main trend in the development of the telecom industry.
- SDN! = OpenFlow
  – SDN - an approach to the separation of the data layer and the management level
  – OpenFlow is one of the implementations. Others, XMPP, SNMP, overlay.

"SDN means thinking differently about networking"

Video about SDN

http://www.youtube.com/watch?v=GRVygzcXrM0