Skoltech

Skolkovo Institute of Science and Technology

Lomonosov Moscow State University

Software-Defined Networks (SDN)

Lecture 4: RUNOS 2.0 OpenFlow Controller

Vasily Pashkov

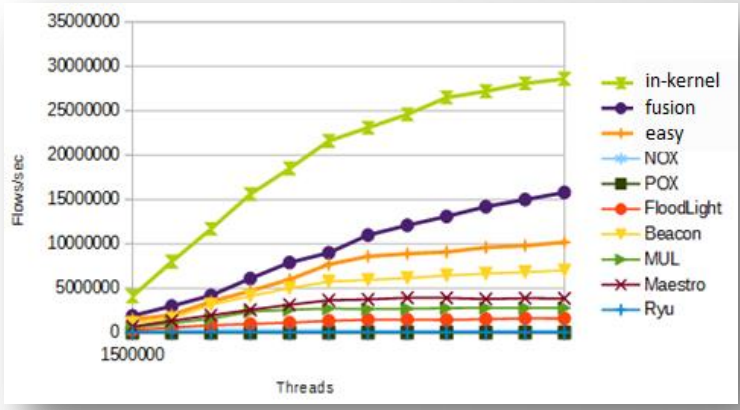
pashkov@lvk.cs.msu.su

RUNOS – Network Operating System

Network Management System - the first Russian SDN controller RUNOS

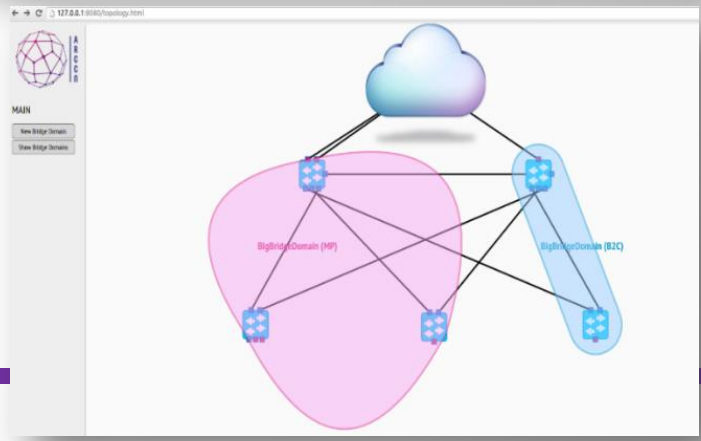
RUssian Network Operation System

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



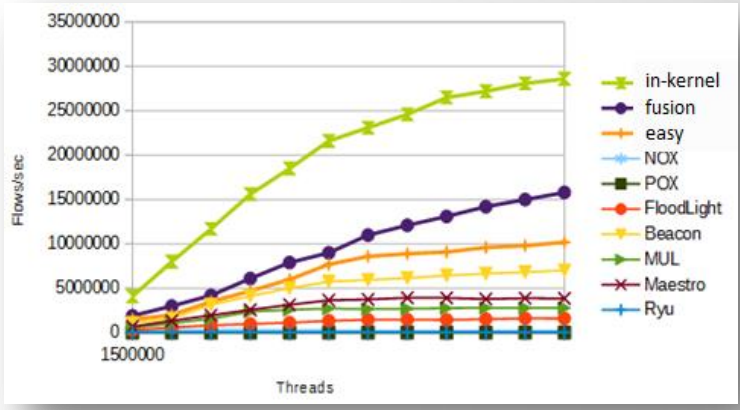
RUNOS – Network Operating System

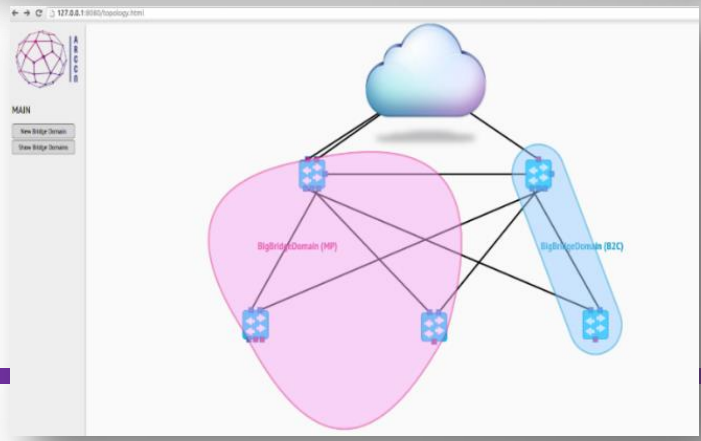
Network Management System - the first Russian SDN controller RUNOS

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





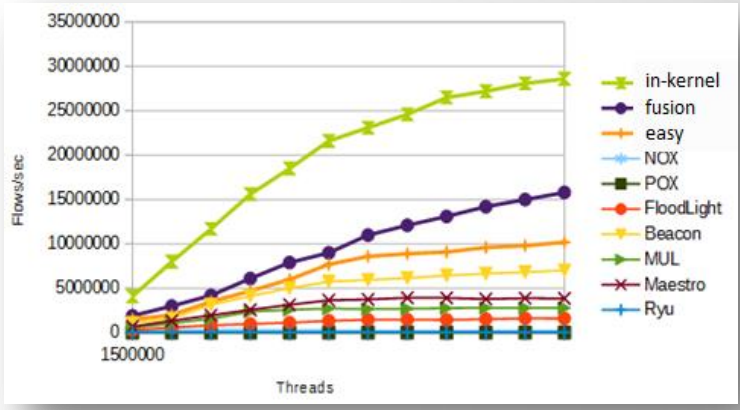
RUNOS – Network Operating System

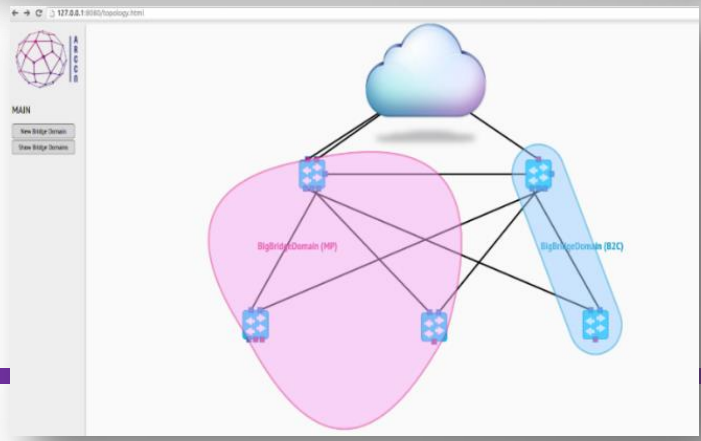
Network Management System - the first Russian SDN controller RUNOS

RUssian Network Operation System

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

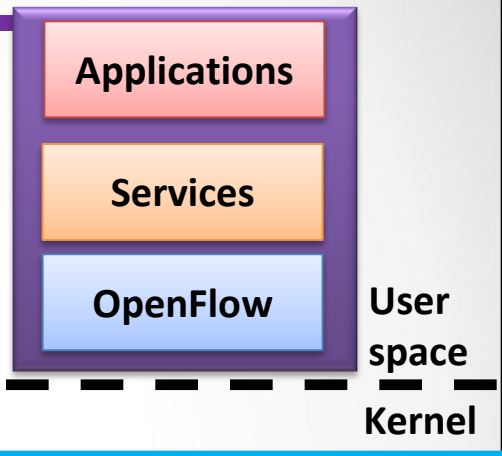
- **Commercial RUNOS version for telecommunications operators**
  - The base is the same as on Github. Customers can develop applications themselves. Learn from accessible materials
  - B2C, B2B services (p2p, mp2mp, multicast, etc.)
  - Active Standby Mode





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



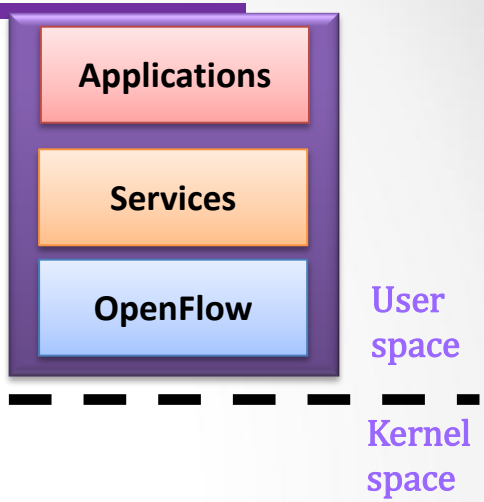
Features:

- 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



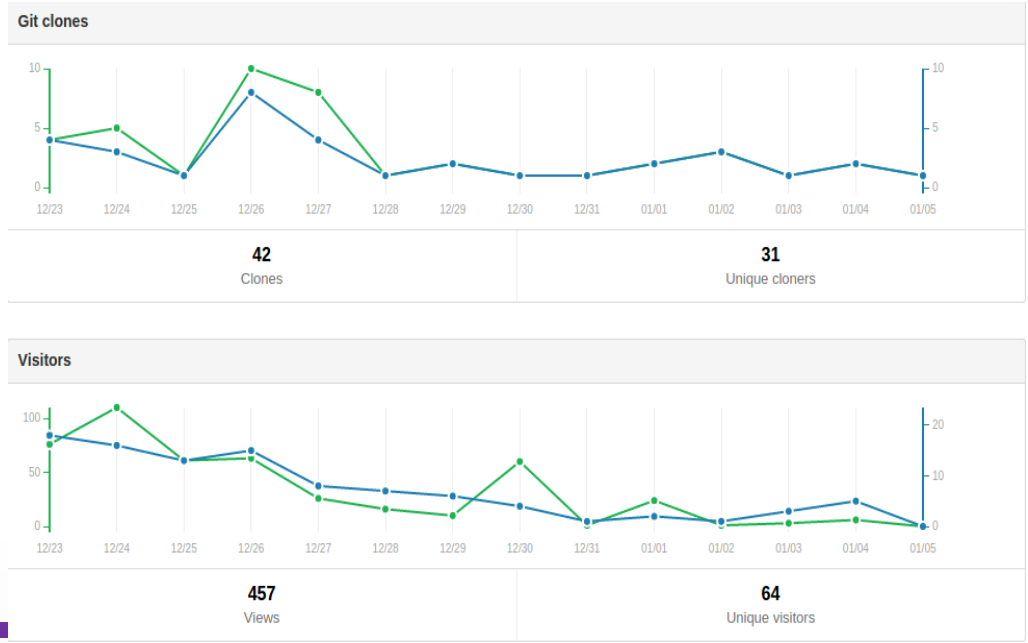
Release Descriptions

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

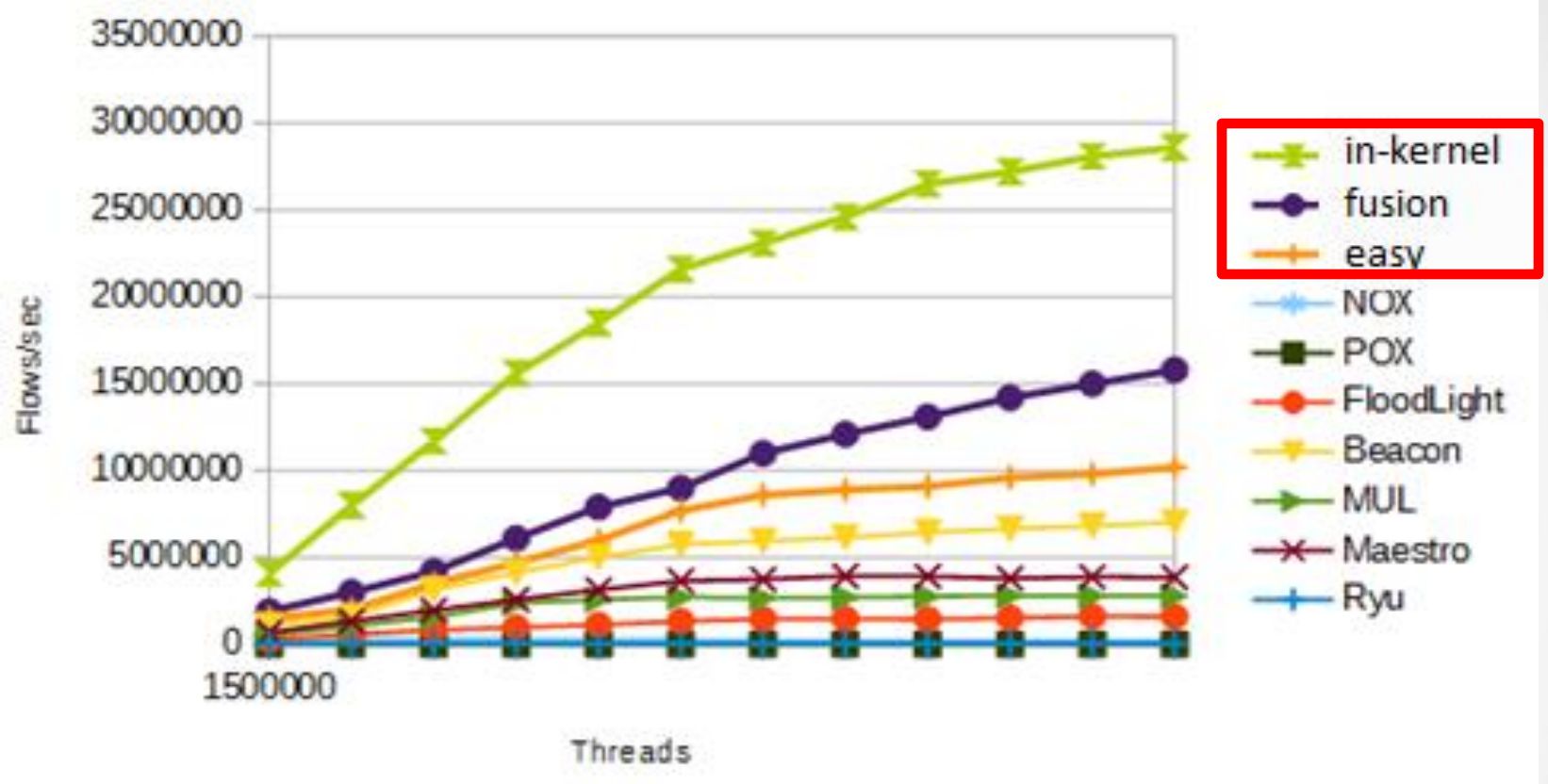
- Sources: http://arccn.github.io/runos/
  - Apache, version 2.0
- Tutorial s(Readme.md + Runos-book)
  - building, installation, running
  - First application tutorial
- Virtual Machine



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**

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):

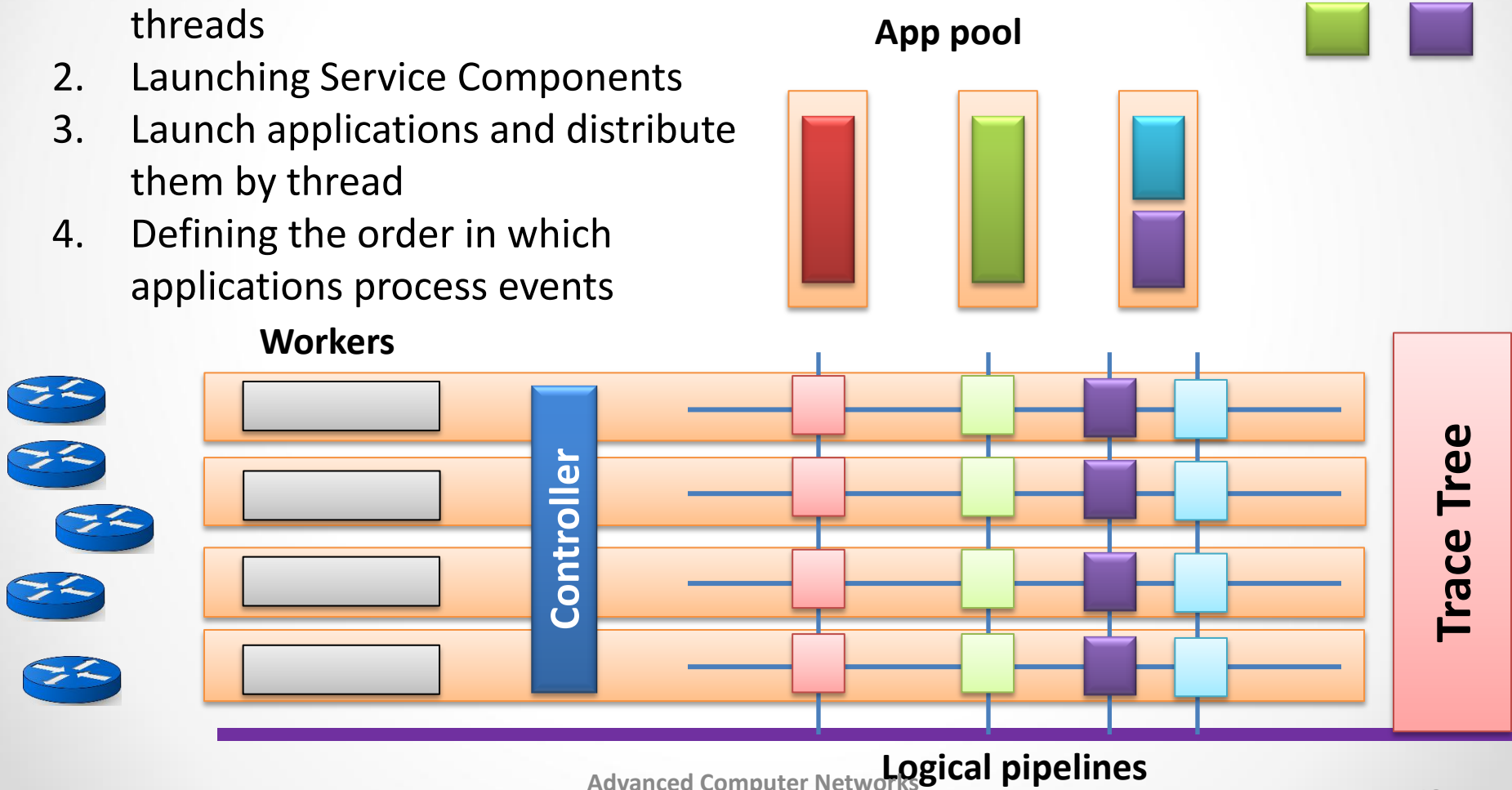
```
“controller”: {
  “threads”: 4
},
“loader”: {
  “threads”: 3
},
“link discovery”: {
   “poll-interval” : 10,
   “pin-to-thread” : 2
},
“learning switch”: {
}
...
```

- **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

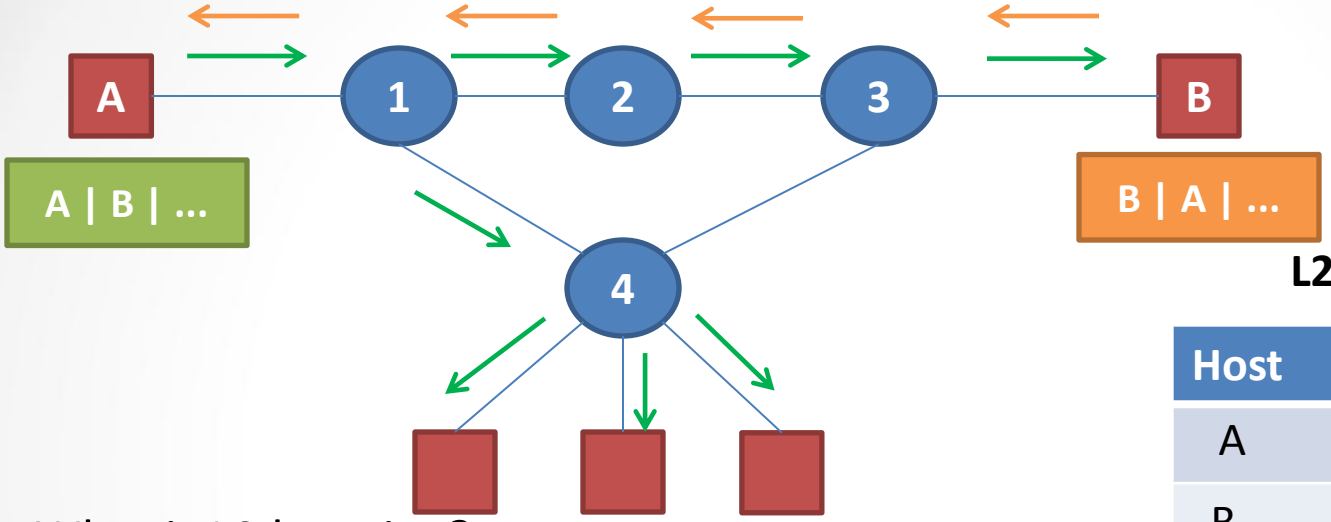
Controller initialization:

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(...)

Skoltech
Skolkovo Institute of Science and Technology

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;
}
```

Skoltech
Skolkovo Institute of Science and Technology

Thanks for your attention!

Vasily Pashkov
pashkov@lvk.cs.msu.su