Building a hierarchical, multi-controller SDN layer to deliver IP routing on OpenFlow 1.x networks with Ryu

Christian Esteve Rothenberg

Agenda

Building a hierarchical, multi-controller SDN layer to deliver IP routing

RouteFlow

- Architecture
- Design and implementation considerations
  - Logic Centralization vs. Physical Distribution
  - Scalability, Reliability, OpenFlow version polyglotism

RFProxy port to Ryu

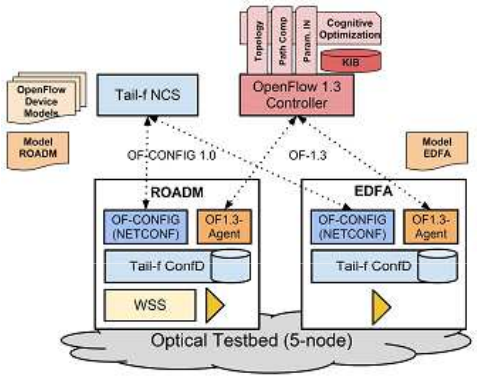
- High-level architecture
- Experiences
- Some benchmarking

Collaboration with University of Campinas

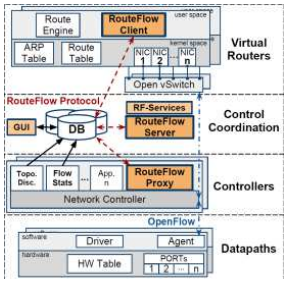
- Ryu OF1.3 use case in a BGP-centric data-center design with TE capabilities.

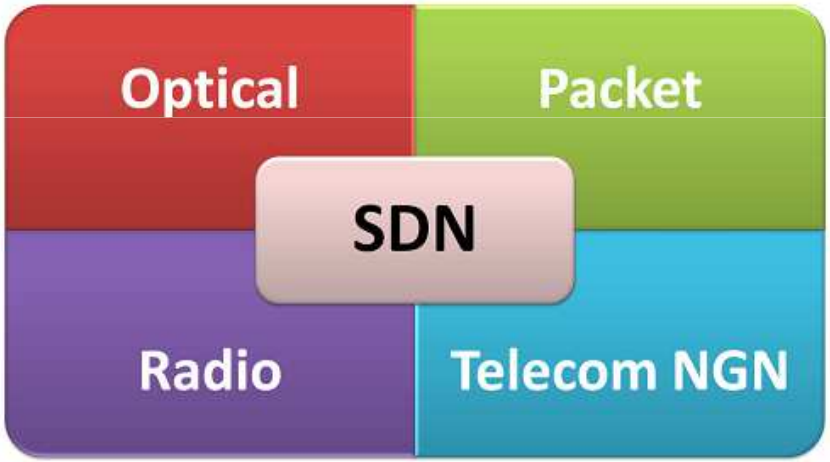
R&D activities with OpenFlow 1.3 and Ryu

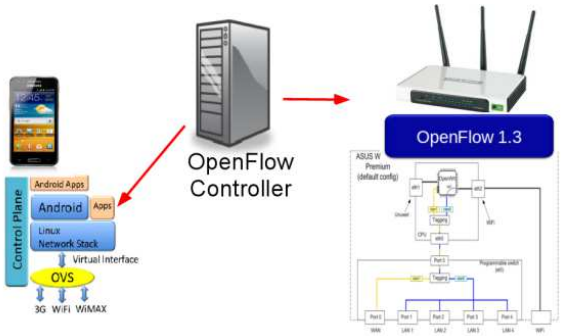
Software-Defined Optical Transport



Software-Defined IP Routing







Software-Defined Wireless Networking

Cloud & Software-Defined Telecom Services

RouteFlow: Introduction

RouteFlow

FUNTTEL

Ministério das Comunicações

GOVERNO FEDERAL BRASIL PAÍS RICO É PAÍS SEM POBREZA

FINEP AGÊNCIA BRASILEIRA DA INOVAÇÃO

Background

Glue of IP routing stacks with OpenFlow

Controller-centric hybrid IP networks

Migration path to SDN

Architecture

Modular (3 components)

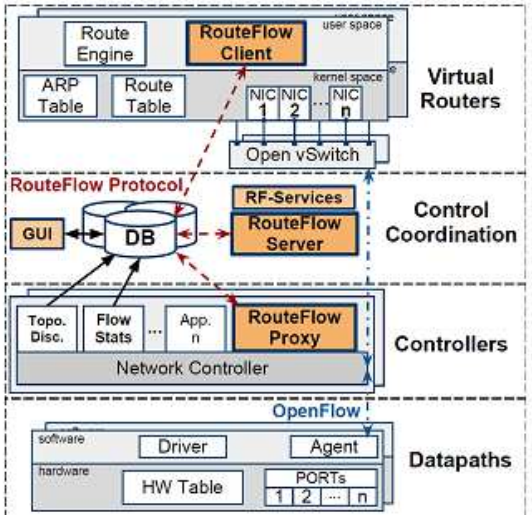
Hierarchical, distributed

Multi-controller support

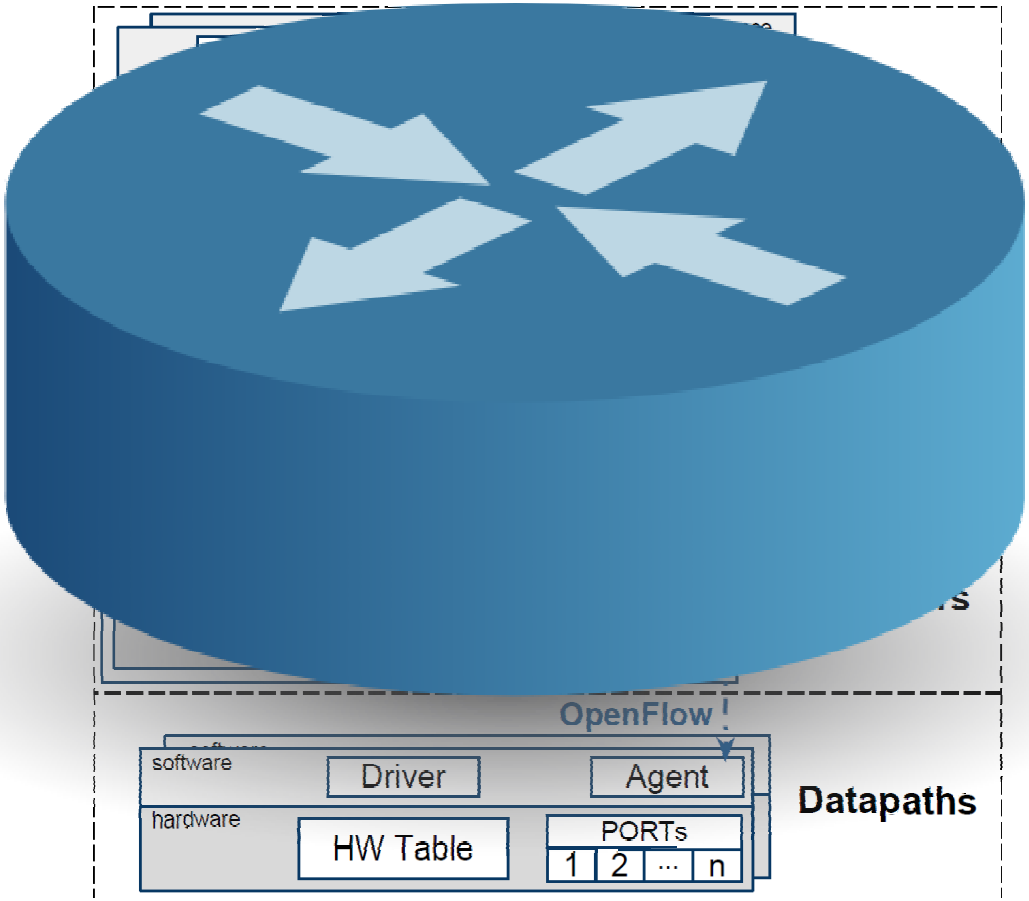
(POX, NOX, Floodlight, Ryu)

Any Linux-based routing stack

(Quagga, XORP, BIRD)



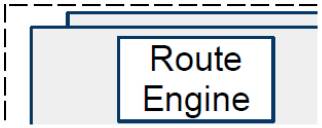
RouteFlow: Basics



Control Plane

Data Plane

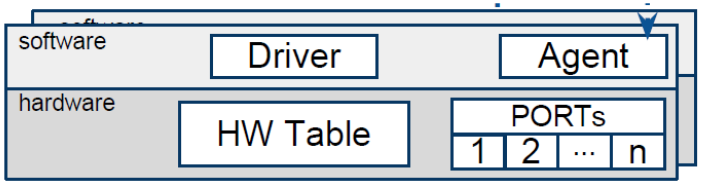
RouteFlow: High-level Architecture



Linux

Control Plane

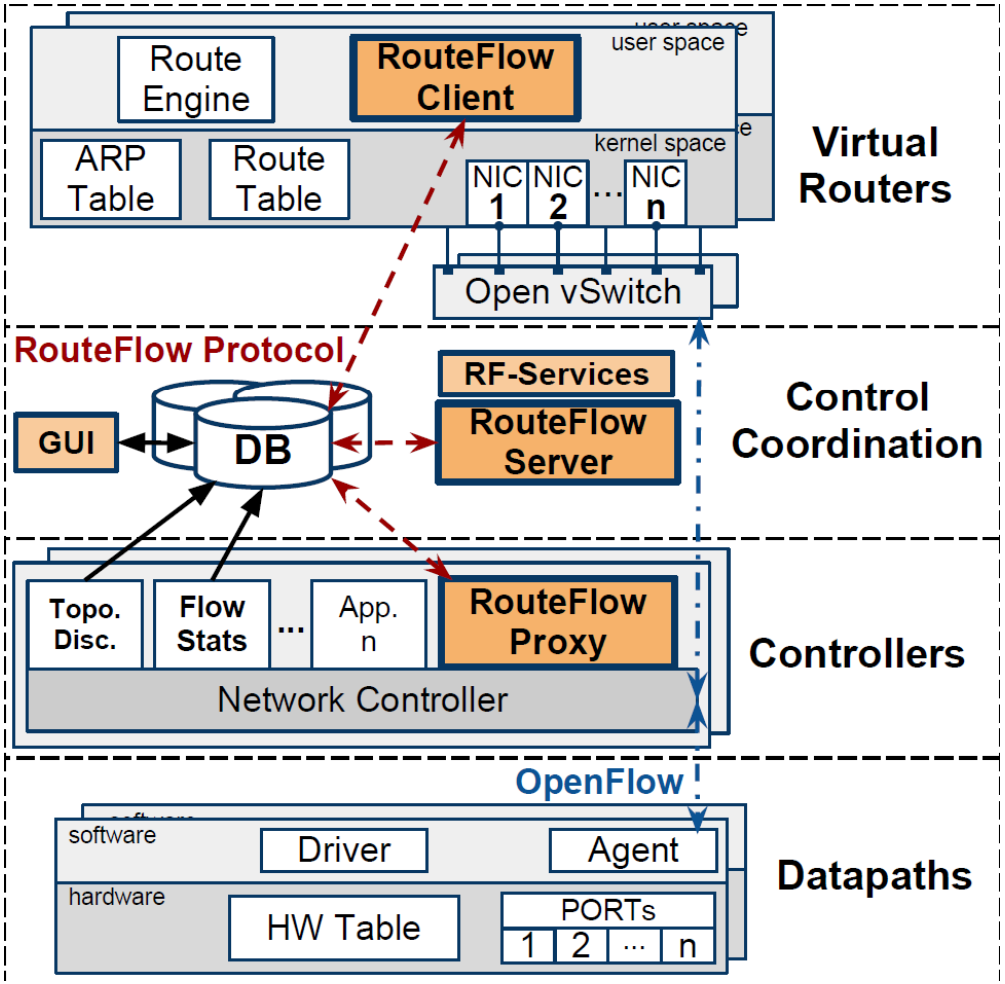
Glue



Datapaths

Data Plane

RouteFlow: High-level Architecture



Control Plane

Glue

OpenFlow

Data Plane

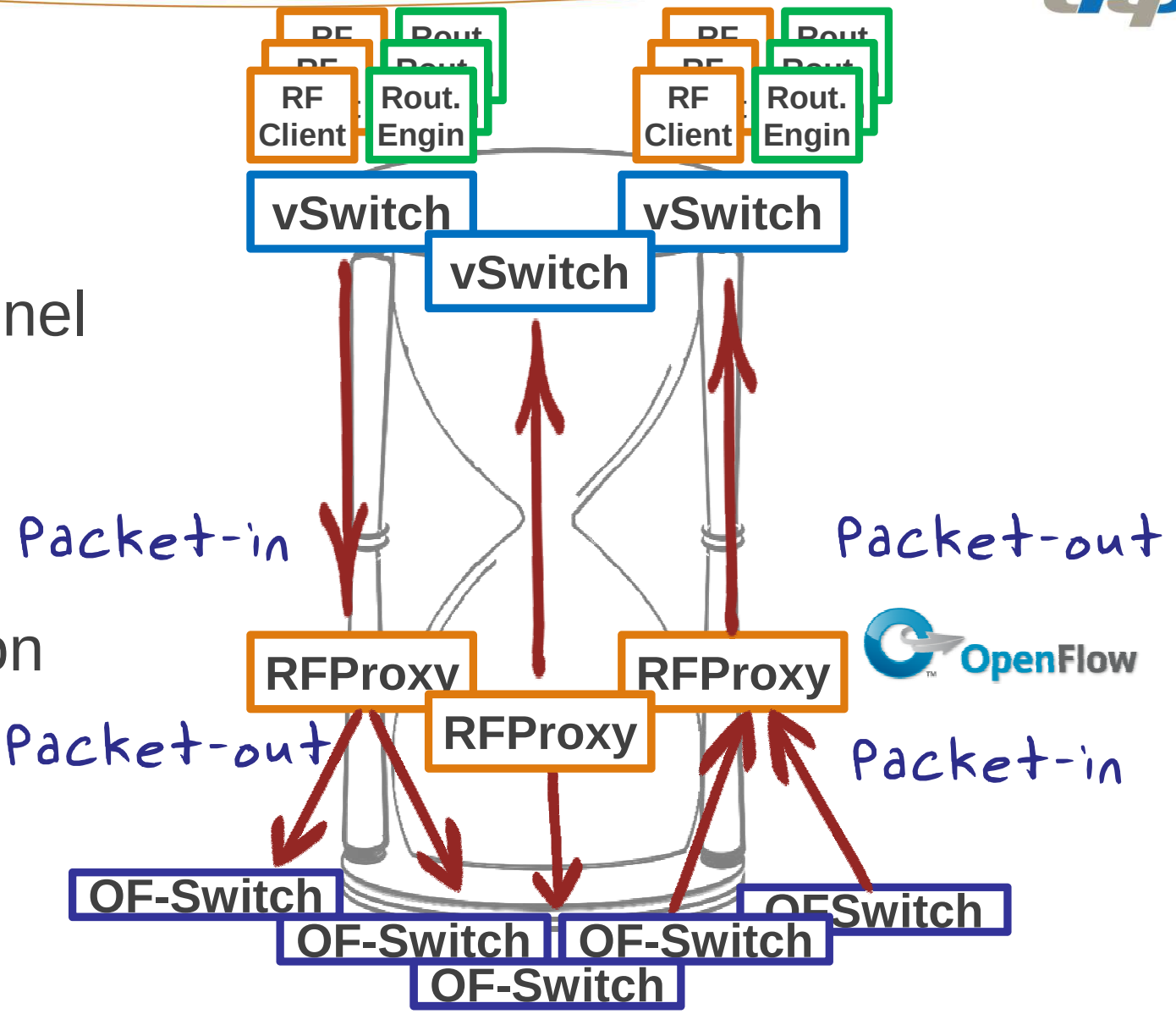
Architectural Discussions

Control-Data Channel

- OpenFlow-based
-

Physical Distribution

- Scalability
- Resiliency



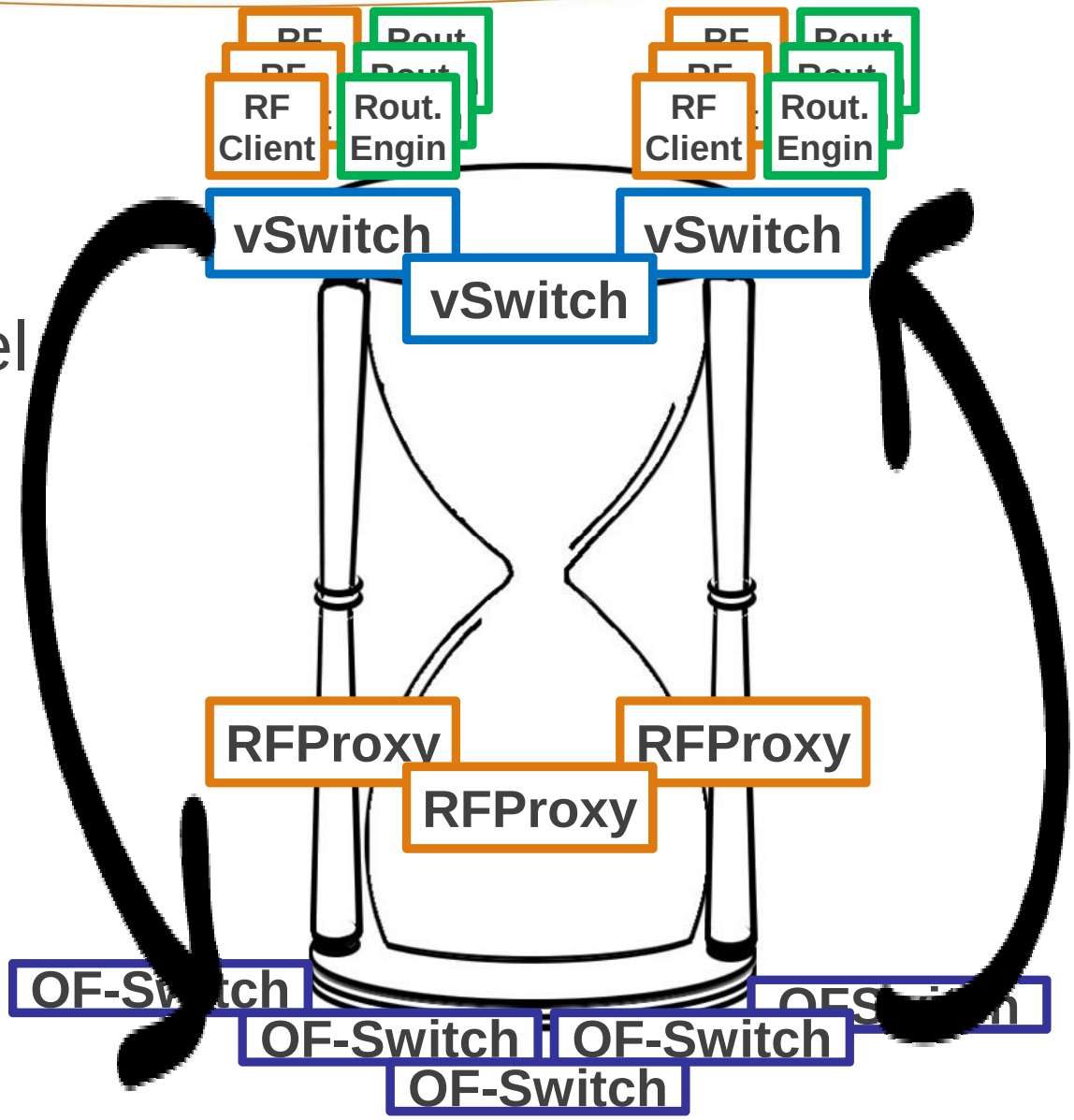
Architectural Discussions

Control-Data Channel

- OpenFlow-based
- OpenFlow-defined

Physical Distribution

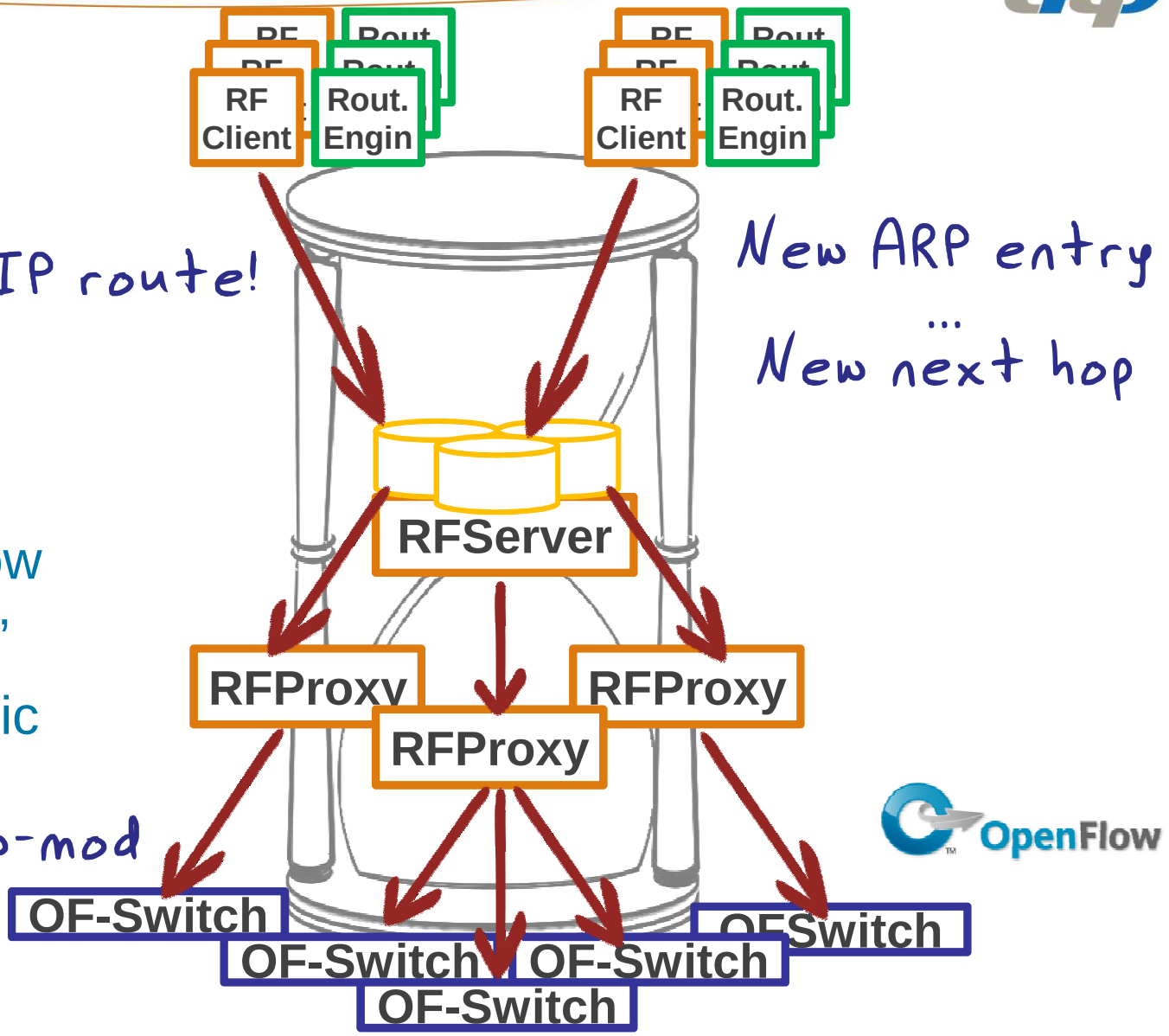
- Scalability
- Resiliency



Architectural Discussions

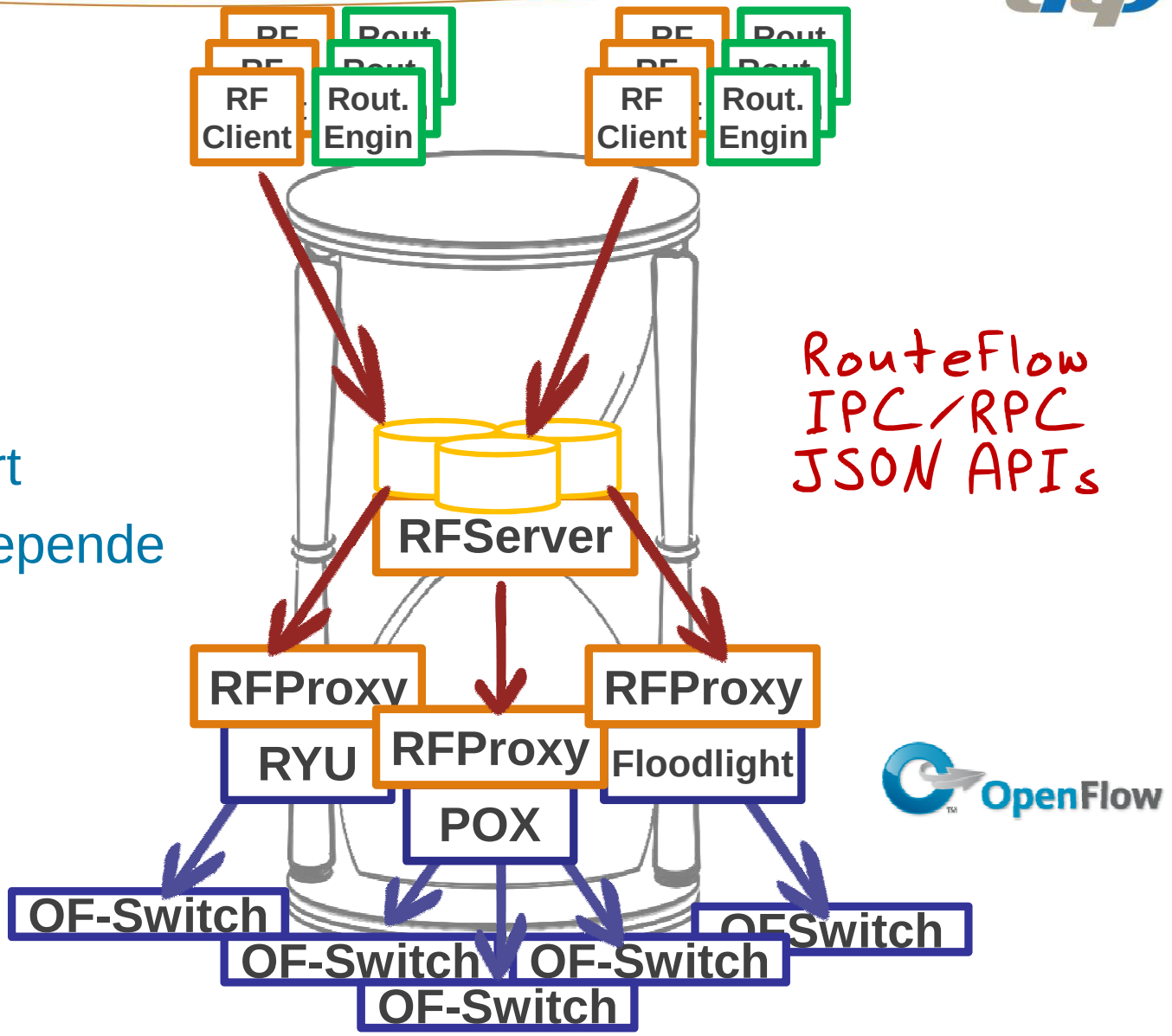
Centralized Logic

- CP/DP Mapping
- RIB-to-FIB-to-OpenFlow
- IP forwarding “policies”
- Intra-domain SDN fabric



Architectural Discussions

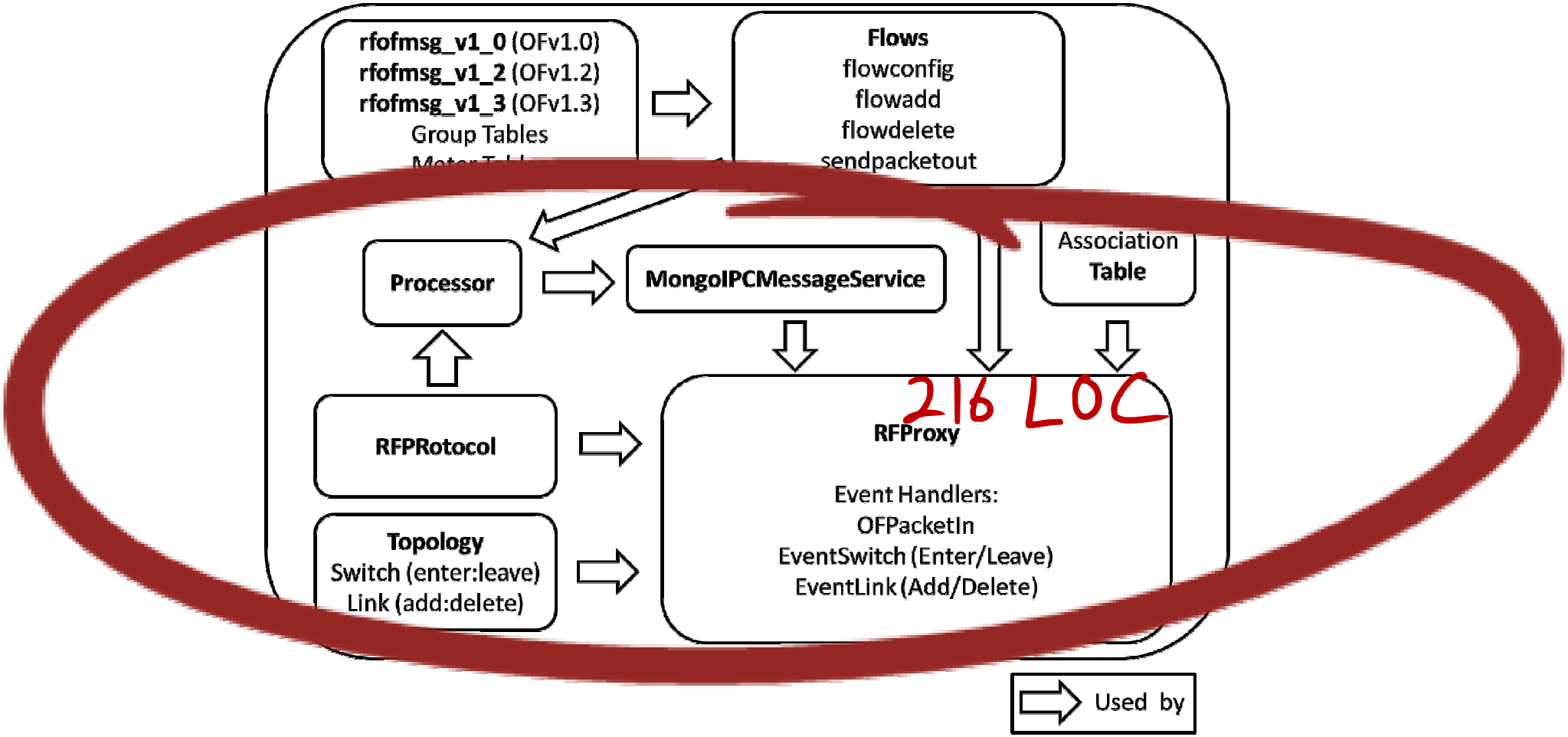
- Hierarchical
- Multi-Controller support
- OpenFlow-version independe



RFProxy app under Ryu with Openflow 1.2 and 1.3

- Ryu v1.8 (Python)
- Simple abstraction through event OpenFlow message handlers
- Uses topology information
- Multipath routing through group tables (OFv1.2 and v1.3)
- QoS through metering tables (OFv1.3)
- Development datapath based on ofsoftswitch 1.x (Ericsson/CPqD)

RFProxy port to Ryu: High-level Architecture



RFProxy port to Ryu: Experience

Easy sintax controller, developer friendly

Support OpenFlow 1.0, 1.2, 1.3

Simple message handlers

- Easy to learn, modify, and build

Recent improvements

- Inter-apps communication

High specialized, helpfull and active developer team:

- Constant upgrades and patchs
- Collaborative development and a lot of tests

RFProxy port to Ryu: Experience

100% feature support for OpenFlow 1.2 and 1.3

REST apps for OpenFlow 1.2 and 1.3

Need more work on 1.2/1.3 API to ease the work with match fields

More constructor options for some classes with default parameters, avoid the need to initialize all parameters (e.g match fields in *flow_mod*)

Ryu OF1.3 use case

Collaboration with University of Campinas

- a BGP-centric data-center design with TE capabilities
- Based on IETF Internet Draft
  “Using BGP for routing in large-scale data centers”
  [draft-lapukhov-bgp-routing-large-dc-02]
- Agreggation of virtual elements following BGP ASN
- Quagga with BGP multipath

Ryu OF1.3 use case

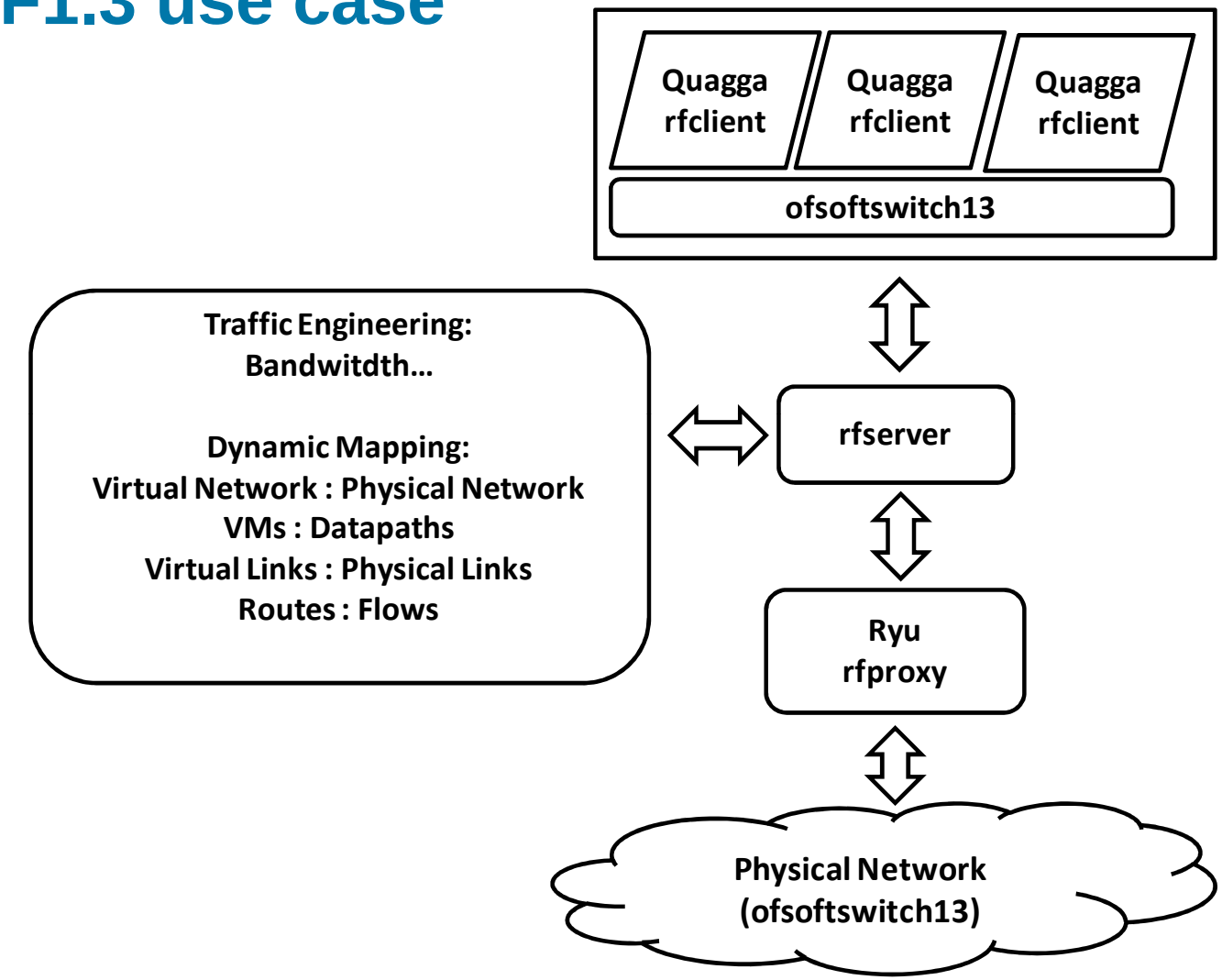
Control plane, RFServer augmented with:

- **Resources:** Define virtual and physical topologies
- **Policies:** paths, bandwidth, isolation, resilience
- **Configuration:** Turn virtual routes into physical flows following policies
- **Allocator:** Check topologies consistency and build flowmod messages

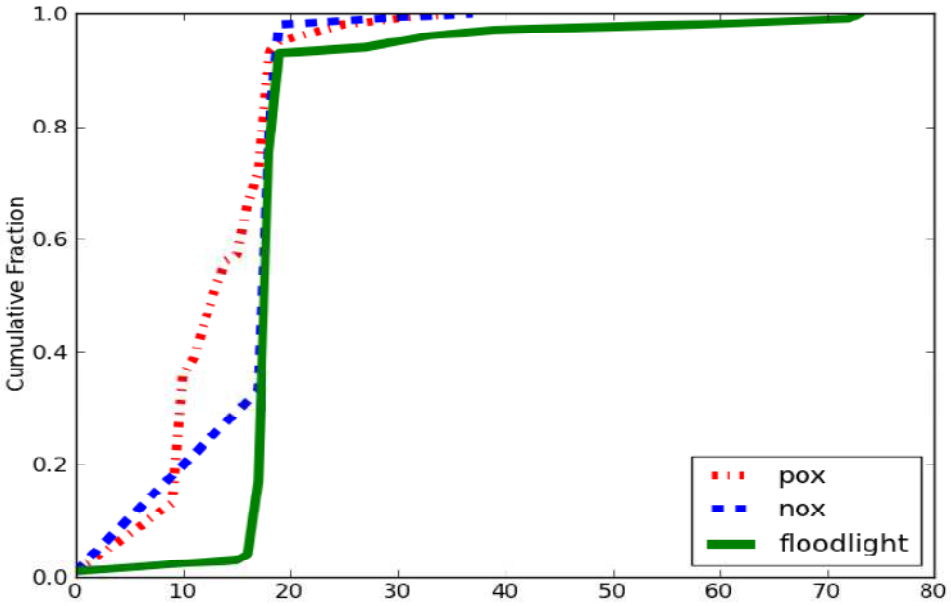
Physical Plane: Data center Clos topology

- Ofsoftswitch1.3 running into Mininet 2.0
- QoS through meter tables: bandwidth mapping
- Multipath through group tables: paths mapping
- Fault-tolerance: NH backup group buckets, master/slave controllers

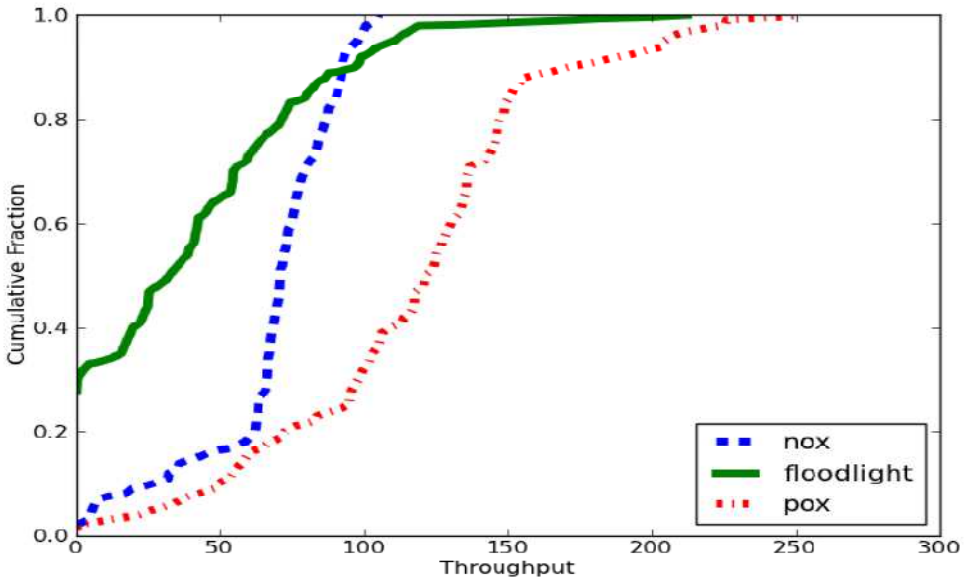
Ryu OF1.3 use case



Some benchmarks

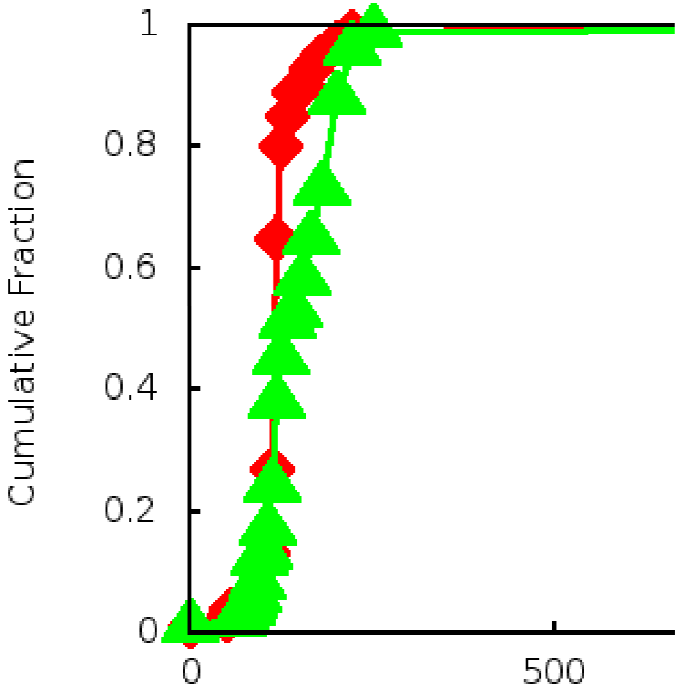


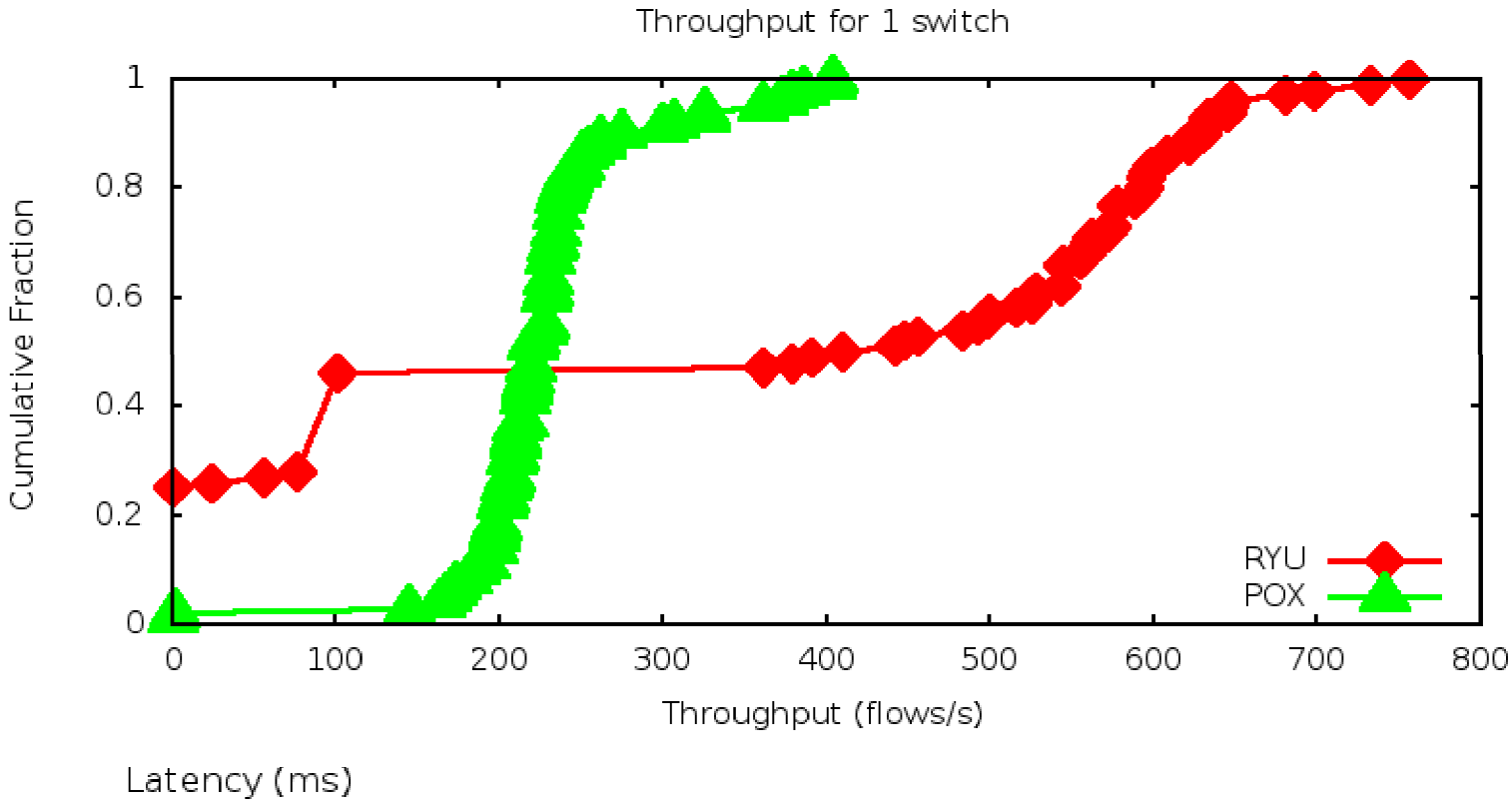
Latency (ms)



Throughput (events/sec)

Some benchmarks





Latency (ms)

Throughput (events/sec)

Acknowledgments

University of Campinas

- Raphael Vicente Rosa (MSc candidate)
- Prof. Edmundo Madeira @ IC/Unicamp

CPqD

- Allan Vidal, Eder Leao… and colleagues

Ericsson

- ofsoftswitch1x developments

RouteFlow

- Community!


Thank You!
Visit our ONS 2013 booth!
www.cpqd.com.br

RFProxy on Ryu (216 LOC)

```
import struct
import logging


import pymongo as mongo


from ryu.app.rflib.ipc.IPC import *
from ryu.app.rflib.ipc.MongoIPC import *
from ryu.app.rflib.ipc.RFProtocol import *
from ryu.app.rflib.openflow.rfofmsg_v1_2 import *
from ryu.app.rflib.ipc.RFProtocolFactory import RFProtocolFactory
from ryu.app.rflib.defs import *


from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import *
from ryu.ofproto import ofproto_v1_2
from ryu.lib.mac import *
from ryu.lib.ip import *
from ryu.lib.dpid import *
from ryu.controller import dpset


log = logging.getLogger('ryu.app.rfproxy')
```

RFProxy on Ryu

```
# Flow installation methods
def flow_config(dp_id, operation_id):
create_config_msg(datapaths.get(dp_id), operation_id)
log.info("ofp_flow_mod(config) was sent to datapath (dp_id=%s)", dpid_to_str(dp_id))


def flow_add(dp_id, address, netmask, src_hwaddress, dst_hwaddress, dst_port):
netmask_int = ipv4_to_int(netmask)
address_int = ipv4_to_int(address)
src_hwaddress_bin = haddr_to_bin(src_hwaddress)
dst_hwaddress_bin = haddr_to_bin(dst_hwaddress)
dp = datapaths.get(dp_id)
conf_flow(dp=dp, ip=address_int, mask=netmask_int, src_hw=src_hwaddress_bin,
dst_hw=dst_hwaddress_bin, dstPort=dst_port, instruction=ADD)
log.info("ofp_flow_mod(add) was sent to datapath (dp_id=%s), (addr=%s), (dst_port=%d)", dpid_to_str(dp_id), address, dst_port)


def flow_delete(dp_id, address, netmask, src_hwaddress):
netmask_int = ipv4_to_int(netmask)
address_int = ipv4_to_int(address)
src_hwaddress_bin = haddr_to_bin(src_hwaddress)
conf_flow(datapaths.get(dp_id), ip=address_int, mask=netmask_int,
src_hw=src_hwaddress_bin, instruction=DEL)
log.info("ofp_flow_mod(del) was sent to datapath (dp_id=%s), (addr=%s)", dpid_to_str(dp_id), address)


conf_flow(datapaths.get(dp_id), ip=address, mask=netmask,
```

RFProxy on Ryu

```
# IPC message Processing
class RFProcessor(IPC.IPCMessageProcessor):
def process(self, from_, to, channel, msg):
type_ = msg.get_type()
if type_ == DATAPATH_CONFIG:
flow_config(msg.get_dp_id(), msg.get_operation_id())
elif type_ == FLOW_MOD:
if (msg.get_is_removal()):
flow_delete(msg.get_dp_id(),
msg.get_address(), msg.get_netmask(),
msg.get_src_hwaddress())
else:
flow_add(msg.get_dp_id(),
msg.get_address(), msg.get_netmask(),
msg.get_src_hwaddress(), msg.get_dst_hwaddress(),
msg.get_dst_port())
if type_ == DATA_PLANE_MAP:
table.update_dp_port(msg.get_dp_id(), msg.get_dp_port(), msg.get_vs_id(), msg.get_vs_port())
return True
```