Software Defined Networking

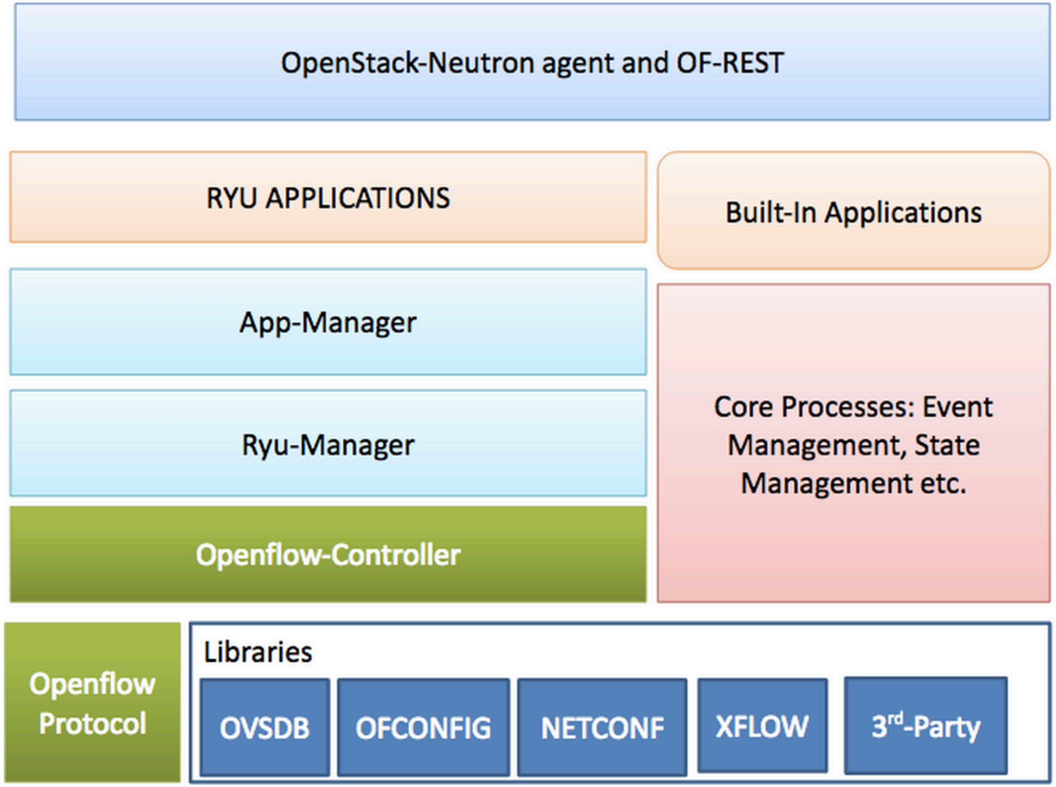
Dr. Nick Feamster
Professor

In this course, you will learn about software defined networking and how it is changing the way communications networks are managed, maintained, and secured.

Commercial Grade Controllers: Ryu

- Overview of Ryu Controller
- API Overview
- Demonstration of Layer 2 Learning Switch

Ryu Architecture



OpenStack-Neutron agent and OF-REST

RYU APPLICATIONS

Built-In Applications

App-Manager

Ryu-Manager

Core Processes: Event Management, State Management etc.

Openflow-Controller

Openflow Protocol

Libraries

OVSDB

OFCONFIG

NETCONF

XFLOW

3rd-Party

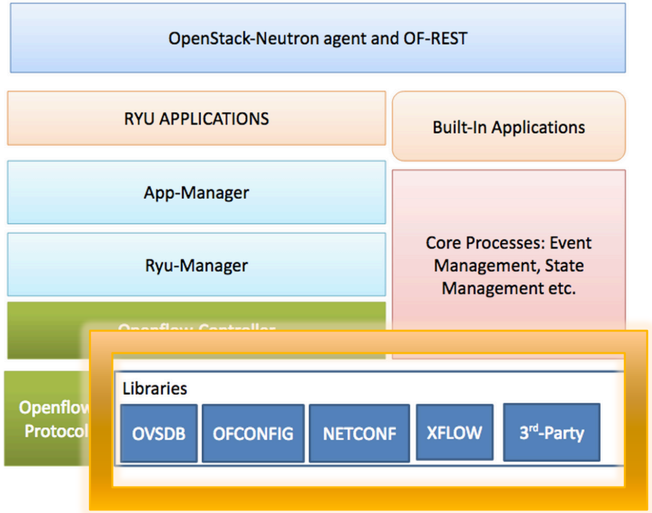
- Libraries
- OpenFlow Controller
- Managers / Core Processes
- Northbound
- Applications

Like other SDN controllers, ability to handle asynchronous events, packet in.

https://github.com/osrg/ryu

Ryu: Libraries

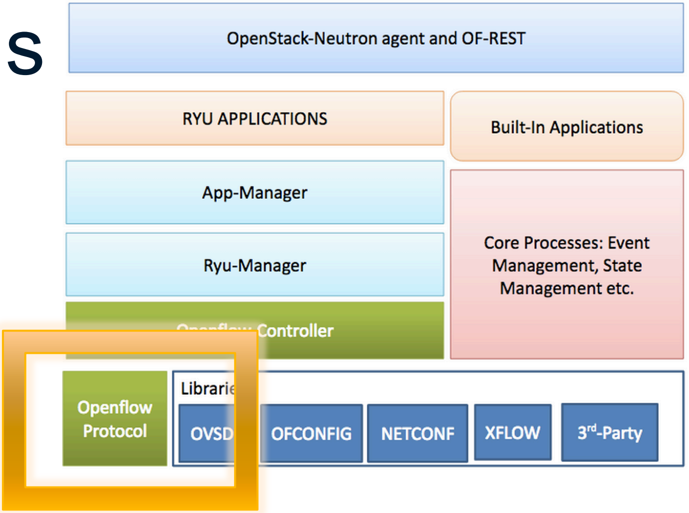
- Support for multiple southbound protocols
  - OF-CONFIG
  - OVSDB
  - NETCONF
  - XFLOW (Netflow, Sflow)
  - Oopen vSwitch Python Binding
- Support for parsing, building various protocol packets



http://goo.gl/TOBwJ4

Ryu: OpenFlow Support

- Support for OpenFlow up to version 1.4
- Controller-to-switch messags
  - Handshake, switch-config, flow-table config, read/modify state, queue config, barrier, …
- Asynchronous messages
  - Packet-in, flow-removed, port-status



OpenStack-Neutron agent and OF-REST

RYU APPLICATIONS

Built-In Applications

App-Manager

Ryu-Manager

Core Processes: Event Management, State Management etc.

Openflow Controller

Openflow Protocol

Librarie

OVSD

OFCONFIG

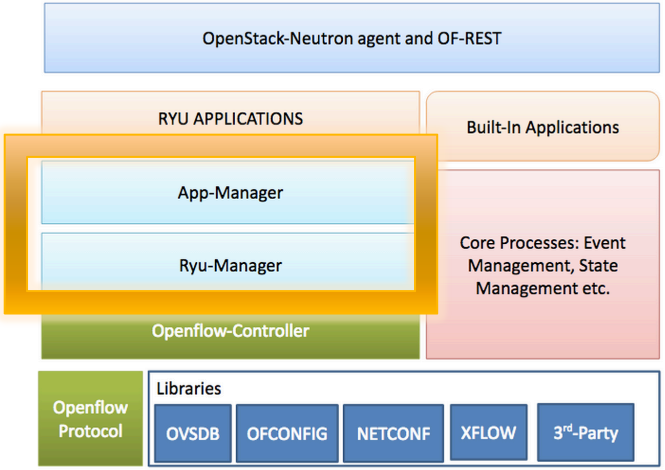
NETCONF

XFLOW

3^{rd}-Party

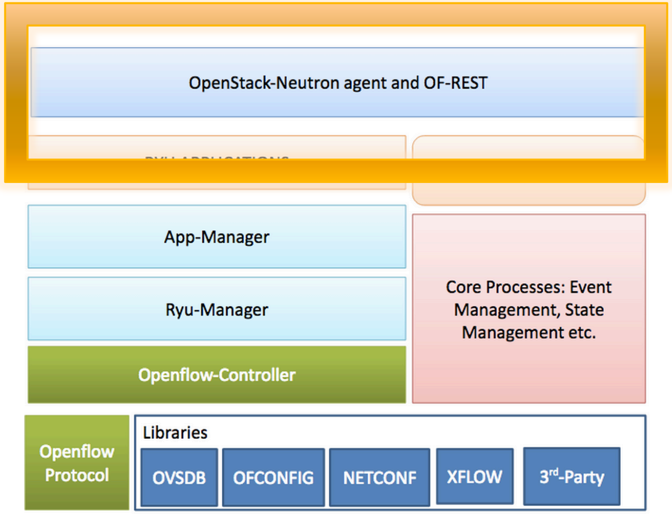
Ryu: Managers and Core-Processes

- Ryu-manager is the main executable
  - Listens to port 6633 by default
  - Any OpenFlow switch can connect to the manager
- All applications inherit from `RyuApp` class



Ryu: Northbound API

- Support for Openstack Neutron
  - Supports GRE-based overlay, VLAN
- REST interface to OpenFlow Operations
- Easy to introduce new REST APIs



OpenStack-Neutron agent and OF-REST

RYU APPLICATIONS

App-Manager

Ryu-Manager

Openflow-Controller

Core Processes: Event Management, State Management etc.

Openflow Protocol

Libraries

OVSDB

OFCONFIG

NETCONF

XFLOW

3rd-Party

Ryu: Applications

- Ryu ships with many applications
  - Simple switch
  - Router
  - Firewall
  - …
- Application has a FIFO event queue
  - Event processing is blocking

Simple Switch Overview

- Simple Python program
- Decorated set_ev_cls function is called for every packet_in

```
from ryu.base import app_manager

class L2Switch(app_manager.RyuApp):
    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)


@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath
        ofp = dp.ofproto
        ofp_parser = dp.ofproto_parser

        actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD)]
        out = ofp_parser.OFPPacketOut(
            datapath=dp, buffer_id=msg.buffer_id,
                                                      in_port=msg.in_port,
                                                      actions=actions)
        dp.send_msg(out)
```

Code Structure

- **app/** – Contains set of applications that run on-top of the controller.
- **base/** – Contains the base class for RYU applications. The RyuApp class in the app_manager.py file is inherited when creating a new application.
- **controller/** – Contains the required set of files to handle OpenFlow functions (e.g., packets from switches, generating flows, handling network events, gathering statistics etc).
- **lib/** – Contains set of packet libraries to parse different protocol headers and a library for OFConfig. In addition, it includes parsers for Netflow and sFlow too.
- **ofproto/** – Contains the OpenFlow protocol specific information and related parsers to support different versions of OF protocol (1.0, 1.2, 1.3, 1.4)
- **topology/** – Contains code that performs topology discovery related to OpenFlow switches and handles associated information (e.g., ports, links etc). Internally uses LLDP.