THE UNIVERSITY OF TEXAS AT EL PASO

Ryu: Network Operating System (NOS)

Jesus Minjares

What is a Network Operating System?

Network Operating System:

Similar to an OS for a computer system

- **Resource management:** device management (host, switches, middle boxes), topology management, bandwidth management
- **Abstraction layer:** hide the details of configuring the network
- **Common services:** shortest path computation, event detection, security services

Controller

- Controllers (Network OSs) can be centralized or distributed
  - **Centralized: Ryu**, Maestro, Beacon, Trema, ProgrammableFlow, Floodlight
  - **Distributed:** ONOS, Onix, HyperFlow, yang

Ryu

- Open-source network operating system
  - Fully written in **Python**
- **Ryu** means “**Dragon**” in Japanese
  - Hence the *dragon logo*

Ryu Logo

Ryu Installation 🐉

Installation

```
# install from pip
pip install ryu

# Update git submodule
git submodule update --init --recursive
cd ryu; pip install .

# Optional Requirements
pip install -r tools/optional-requires

# Prerequisites
# install as super user
sudo apt install gcc python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev
```

Git Submodule

```
[submodule "ryu"]
    path = ryu
    url = https://github.com/faucetsdn/ryu.git
```

Getting Started with Ryu 🐉

How to run Ryu Application

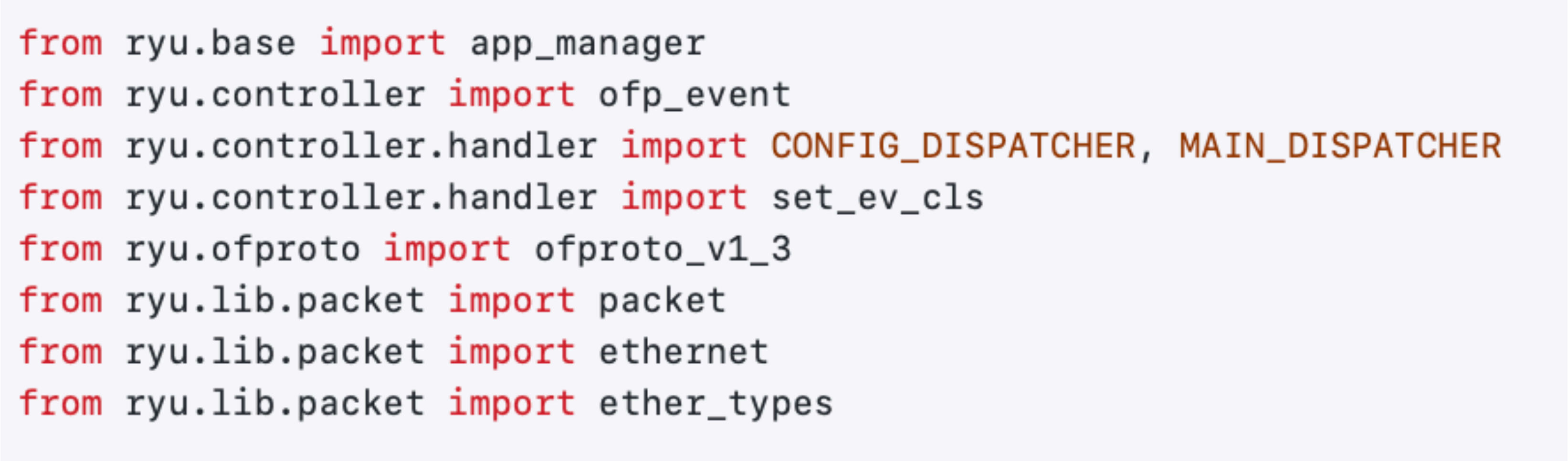
Ryu was design to run with the same concept as python scripts, instead of using ***python*** we would use ***ryu-manager***.

ryu-manager is the executable for Ryu applications. **ryu-manager** loads Ryu applications and run it.

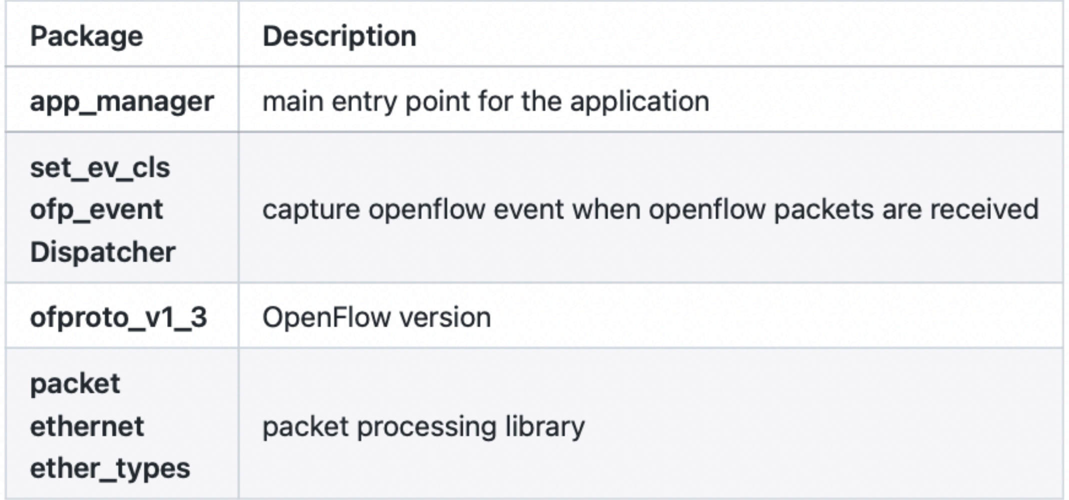
```
ryu-manager <application file name>
```

Creating Applications with Ryu

Step 1: Import packages



```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
```



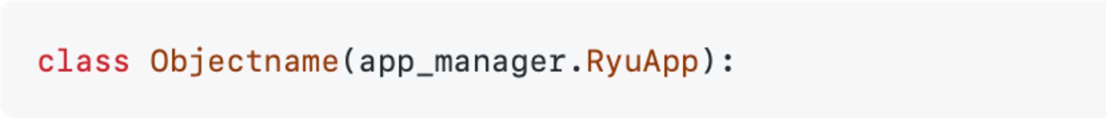
Package	Description
app_manager	main entry point for the application
set_ev_cls **ofp_event** **Dispatcher**	capture openflow event when openflow packets are received
ofproto_v1_3	OpenFlow version
packet **ethernet** **ether_types**	packet processing library

Table 1 Description of the Ryu packages

Creating Applications with Ryu Cont.

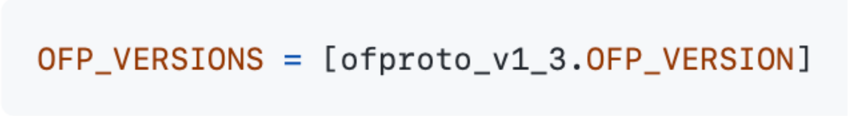
Step 2: Create Class

When creating an ryu application class, always pass the app_manager.RyuApp.



```
class Objectname(app_manager.RyuApp):
```

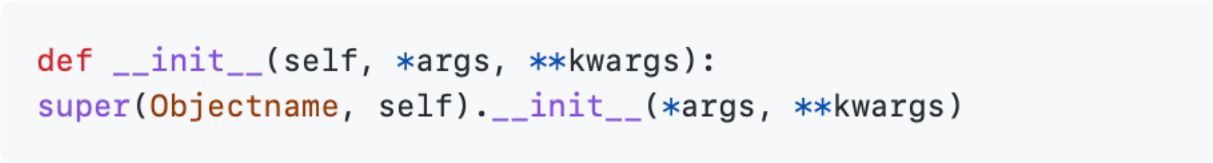
Step 3: Set the OpenFlow version



```
OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
```

Step 4: Define class constructor

When creating the constructor always user ***super*** to inheritance Ryu base class



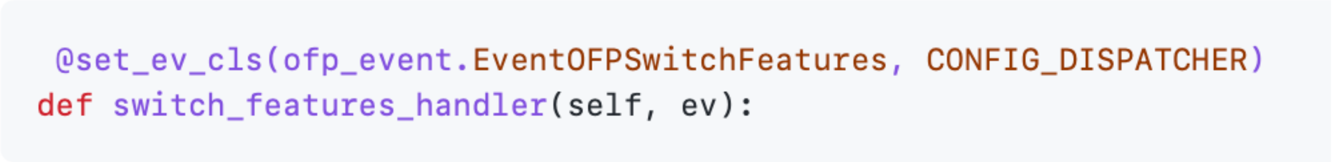
```
def __init__(self, *args, **kwargs):
super(Objectname, self).__init__(*args, **kwargs)
```

Creating Applications with Ryu Cont.

Step 5: Add Events

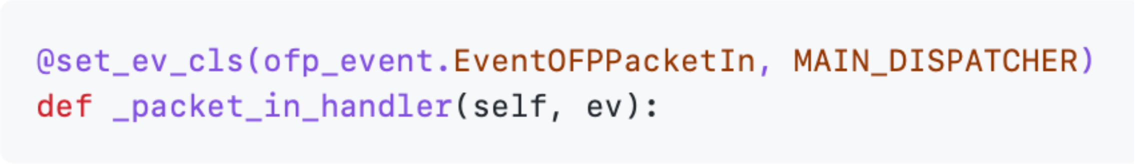
In a Ryu controller, you could add events that you would want to listen too. With the use of decocrat
python we can add functionality to the controller: @set_ev_cls.

The code below will be capture for any event at the OpenFlow switch.



```
 @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
```

we will add another event to listen to the packets that are being received. With the same functionalit
previous event, this _packet_in_handler function will be trigger once packets are being captured.



```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

Creating Applications with Ryu Cont.🐉

Step 6: Ryu Application

ryu-manager <application file name>

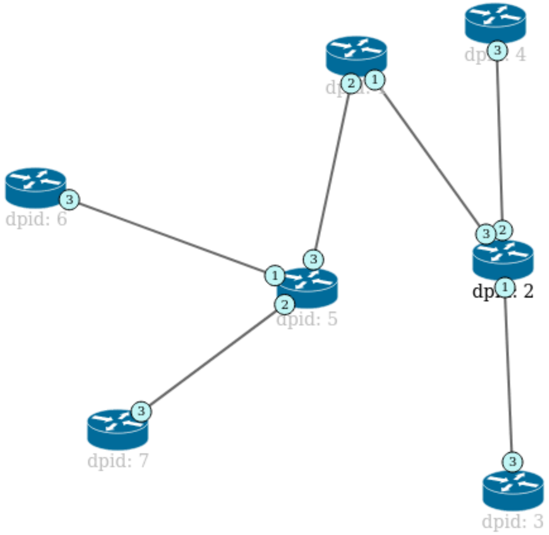
Step 7: Mininet network

```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24
--switch=ovsk,protocols=OpenFlow13 –topo=tree,depth=3,fanout=4
```

Ryu Topology

Ryu has the functionality of creating graphs for the topology

- Provides visual representation of the network



How to run topology functionality:

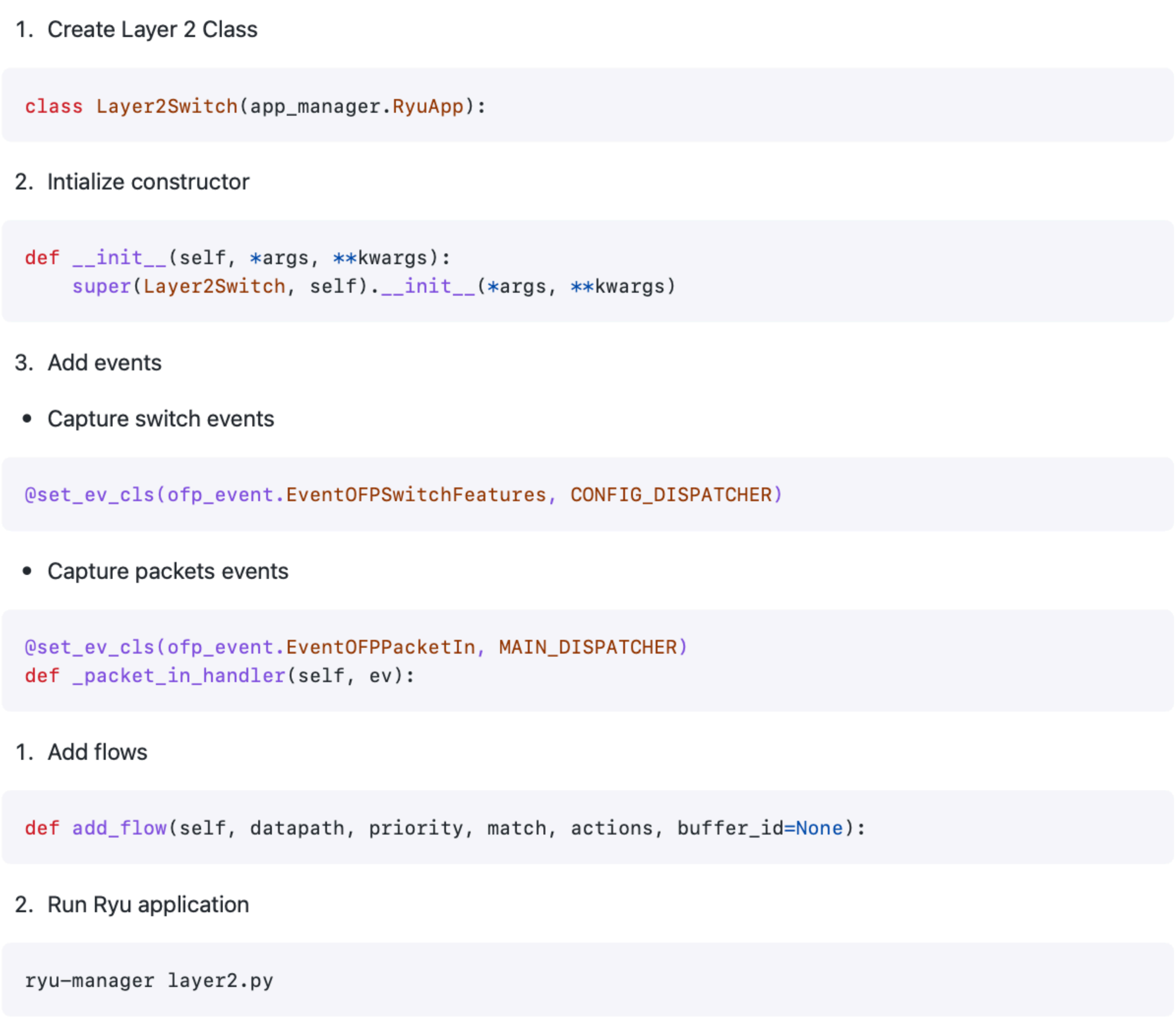
1. Run Ryugui_topology.py

```
ryu run --observe-links ryu/app/gui_topology/gui_topology.py
```

2. Have the mininet topology that you want to generate the graph

3. Open http://<ip_addressofryucontroller>/8080

Creating Layer 2 Switch in Ryu



1. Create Layer 2 Class

```
class Layer2Switch(app_manager.RyuApp):
```

2. Intialize constructor

```
def __init__(self, *args, **kwargs):
    super(Layer2Switch, self).__init__(*args, **kwargs)
```

3. Add events

- Capture switch events

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
```

- Capture packets events

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

1. Add flows

```
def add_flow(self, datapath, priority, match, actions, buffer_id=None):
```

2. Run Ryu application

```
ryu-manager layer2.py
```

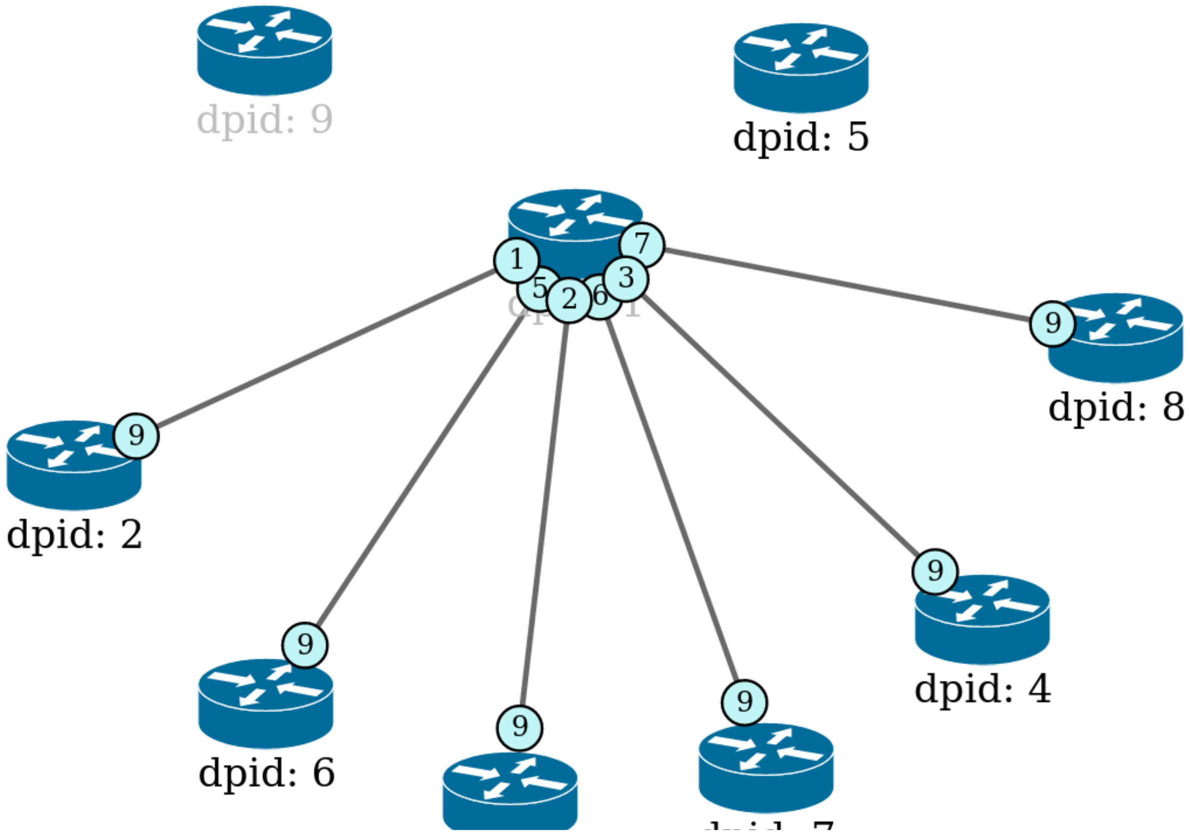
Creating Layer 2 Switch in Ryu Contd.

Run mininet network

```
chmod +x layer2.sh
./layer.sh
```

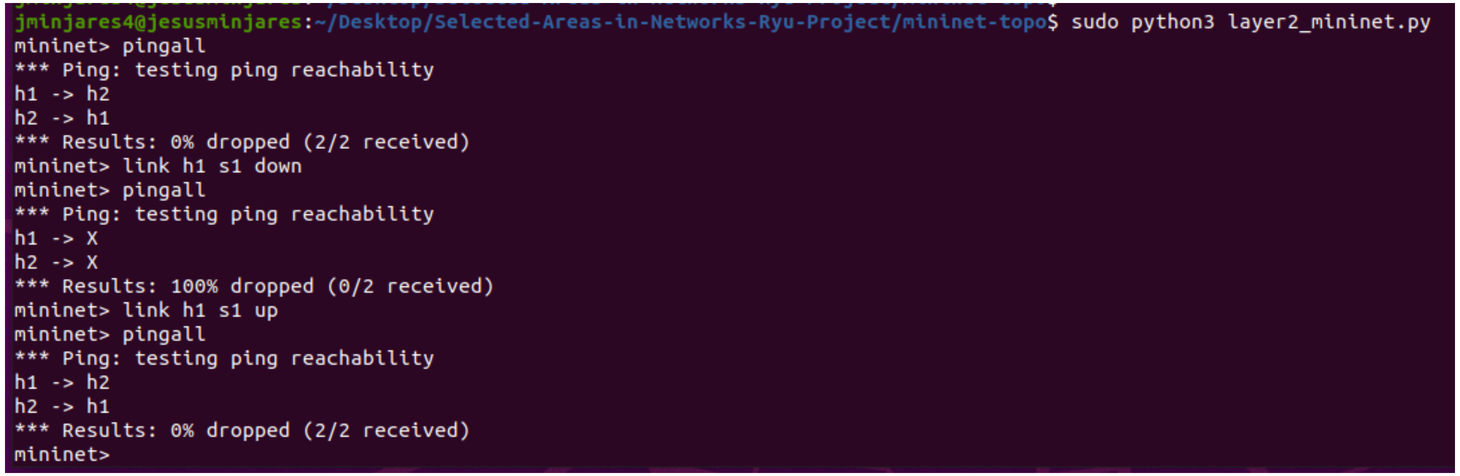
Get topology with

```
chmod +x topology.sh
./topology.sh
```



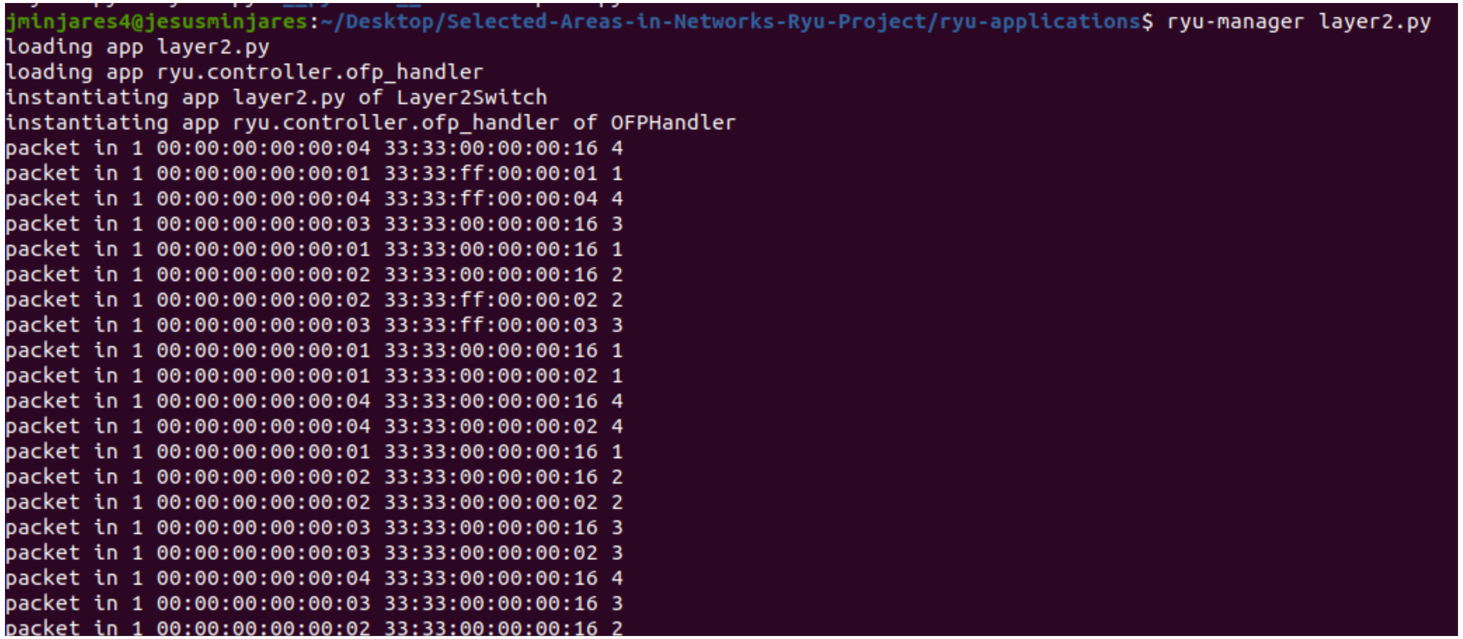
layer2.sh topology using Ryu Topology application

Creating Layer 2 Switch in Ryu Contd.



```
jminjares4@jesusminjares:~/Desktop/Selected-Areas-in-Networks-Ryu-Project/mininet-topo$ sudo python3 layer2_mininet.py
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> link h1 s1 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
mininet> link h1 s1 up
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

Mininet network script output



```
jminjares4@jesusminjares:~/Desktop/Selected-Areas-in-Networks-Ryu-Project/ryu-applications$ ryu-manager layer2.py
loading app layer2.py
loading app ryu.controller.ofp_handler
instantiating app layer2.py of Layer2Switch
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 00:00:00:00:00:04 33:33:00:00:00:16 4
packet in 1 00:00:00:00:00:01 33:33:ff:00:00:01 1
packet in 1 00:00:00:00:00:04 33:33:ff:00:00:04 4
packet in 1 00:00:00:00:00:03 33:33:00:00:00:16 3
packet in 1 00:00:00:00:00:01 33:33:00:00:00:16 1
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2
packet in 1 00:00:00:00:00:02 33:33:ff:00:00:02 2
packet in 1 00:00:00:00:00:03 33:33:ff:00:00:03 3
packet in 1 00:00:00:00:00:01 33:33:00:00:00:16 1
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
packet in 1 00:00:00:00:00:04 33:33:00:00:00:16 4
packet in 1 00:00:00:00:00:04 33:33:00:00:00:02 4
packet in 1 00:00:00:00:00:01 33:33:00:00:00:16 1
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
packet in 1 00:00:00:00:00:03 33:33:00:00:00:16 3
packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 3
packet in 1 00:00:00:00:00:04 33:33:00:00:00:16 4
packet in 1 00:00:00:00:00:03 33:33:00:00:00:16 3
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2
```

Ryu Layer 2 Switch application output

GitHub Documentation

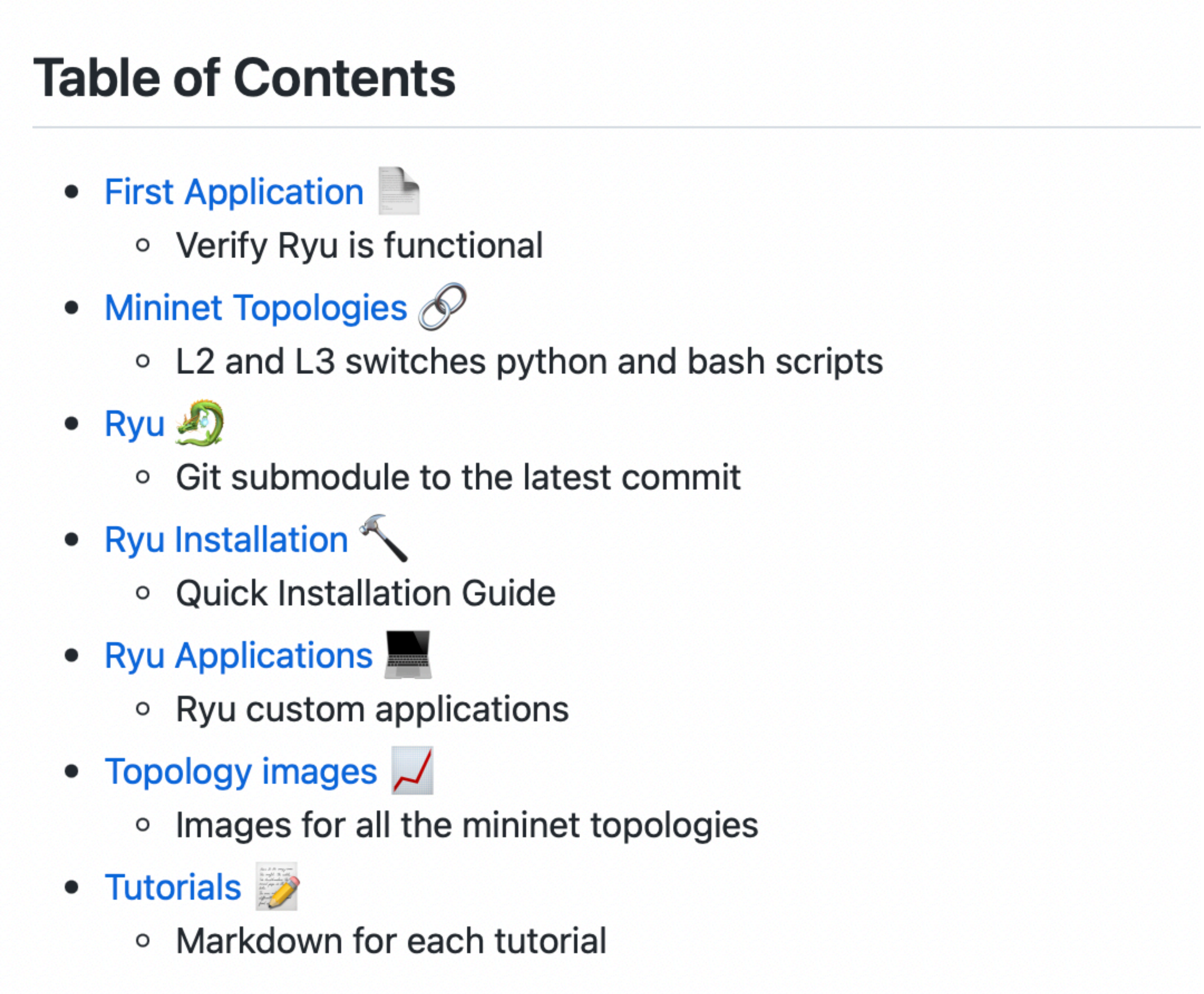


Table of Contents

- First Application 📄
  - Verify Ryu is functional
- Mininet Topologies 🔗
  - L2 and L3 switches python and bash scripts
- Ryu 🐉
  - Git submodule to the latest commit
- Ryu Installation 🔨
  - Quick Installation Guide
- Ryu Applications 💻
  - Ryu custom applications
- Topology images 📈
  - Images for all the mininet topologies
- Tutorials 📝
  - Markdown for each tutorial

Ryu-Project-Jesus-Minjares

References

Ryu Open source

- Ryu API
- Ryu GitHub

Project Documentation

- Ryu-Project: Jesus Minjares