Faucet Documentation

Release

Faucet Developers

Dec 04, 2017

Contents

CHAPTER 1

User Documentation

1.1 Introduction to Faucet

1.1.1 What is Faucet?

1.1.2 What is Gauge?

1.1.3 Why Faucet?

https://queue.acm.org/detail.cfm?id=3015763

1.1.4 Getting Help

We use maintain a number of mailing lists for communicating with users and developers:

- faucet-announce
- faucet-dev
- faucet-users

We also have the #faucetsdn IRC channel on freenode.

A few tutorial videos are available on our YouTube channel.

The faucetsdn blog and faucetsdn twitter are good places to keep up with the latest news about faucet.

If you find bugs, or if have feature requests, please create an issue on our bug tracker.

1.2 Installation

1.2.1 Common Installation Tasks

These tasks are required by all installation methods.

You will need to provide an initial configuration files for FAUCET and Gauge, and create directores for FAUCET and Gauge to log to.

```
mkdir -p /etc/ryu/faucet
mkdir -p /var/log/ryu/faucet
mkdir -p /var/log/ryu/gauge
$EDITOR /etc/ryu/faucet/faucet.yaml
$EDITOR /etc/ryu/faucet/gauge.yaml
```

This example `faucet.yaml` file creates an untagged VLAN between ports 1 and 2 on DP 0x1. See *Configuration* for more advanced configuration. See *Vendor-specific Documentation* for how to configure your switch.

```
vlans:
    100:
        name: "dev VLAN"
dps:
    switch-1:
        dp_id: 0x1
        interfaces:
            1:
                native_vlan: 100
            2:
                native_vlan: 100
```

This example `gauge.yaml` file instructs Gauge to poll the switch at 10s intervals and store the results in InfluxDB. See *Configuration* for more advanced configuration.

```
faucet_configs:
    - '/etc/ryu/faucet/faucet.yaml'
watchers:
    port_stats:
        dps: ['switch-1']
        type: 'port_stats'
        interval: 10
        db: 'influx'
    port_state:
        dps: ['switch-1']
        type: 'port_state'
        interval: 10
        db: 'influx'
dbs:
    influx:
        type: 'influx'
        influx_db: 'faucet'
        influx_host: '172.17.0.1'
        influx_port: 8086
        influx_user: 'faucet'
        influx_pwd: ''
        influx_timeout: 10
```

1.2.2 Encrypted Control Channel

This section outlines the steps needed to test that a switch supports self-signed certificates for TLS based Openflow connections.

Prepare the keys and certificates

Generate key pairs for the controller.

```
/usr/bin/openssl genrsa -out /etc/ryu/ctrlr.key 2048
/usr/bin/openssl req -new -x509 -nodes -days 3650 -subj '/C=US/ST=CA/L=Mountain View/
↪O=Faucet/OU=Faucet/CN=CTRLR_1' -key /etc/ryu/ctrlr.key -out /etc/ryu/ctrlr.cert
```

Generate key pairs for the switch.

```
/usr/bin/openssl genrsa -out /etc/ryu/sw.key 2048
/usr/bin/openssl req -new -x509 -nodes -days 3650 -subj '/C=US/ST=CA/L=Mountain View/
↪O=Faucet/OU=Faucet/CN=SW_1' -key /etc/ryu/sw.key -out /etc/ryu/sw.cert
```

Push key pairs to the switch

Copy `/etc/ryu/ctrlr.cert` `/etc/ryu/sw.key` and `/etc/ryu/sw.cert` to the switch. Configure the switch to use the keys.

For example, the command for OVS would be:

```
ovs-vsctl set-ssl  /etc/ryu/sw.key /etc/ryu/sw.cert  /etc/ryu/ctrlr.cert
ovs-vsctl set-controller br0 ssl:<ctrlr_ip>:6653
```

Start Faucet with the keys (make sure the keys are readable by the user that starts the faucet process)

```
ryu-manager --ctl-privkey /etc/ryu/ctrlr.key --ctl-cert /etc/ryu/ctrlr.cert  --ca-
↪certs /etc/ryu/sw.cert faucet.faucet --verbose
```

Support multiple switches

To support multiple switches, generate key pairs for each switch, and concatenate their certificates into one file and use that file as */etc/ryu/sw.cert*.

1.2.3 Installation with Docker on Ubuntu with systemd

We provide official automated builds on Docker Hub so that you can easily run Faucet and it's components in a self-contained environment without installing on the main host system.

See *Docker* for how to install the FAUCET and Gauge images.

You can configure systemd to start the containers automatically:

```
$EDITOR /etc/systemd/system/faucet.service
$EDITOR /etc/systemd/system/gauge.service
systemctl daemon-reload
systemctl enable faucet.service
systemctl enable gauge.service
```

```
systemctl restart faucet
systemctl restart gauge
```

`/etc/systemd/system/faucet.service` should contain:

```
[Unit]
Description="FAUCET OpenFlow switch controller"
After=network-online.target
Wants=network-online.target
After=docker.service

[Service]
Restart=always
ExecStart=/usr/bin/docker start -a faucet
ExecStop=/usr/bin/docker stop -t 2 faucet

[Install]
WantedBy=multi-user.target
```

`/etc/systemd/system/gauge.service` should contain:

```
[Unit]
Description="Gauge OpenFlow switch controller"
After=network-online.target
Wants=network-online.target
After=docker.service

[Service]
Restart=always
ExecStart=/usr/bin/docker start -a gauge
ExecStop=/usr/bin/docker stop -t 2 gauge

[Install]
WantedBy=multi-user.target
```

You can check that FAUCET and Gauge are running via systemd or via docker:

```
service faucet status
service gauge status
docker ps
```

1.2.4 Installation with pip

You can install the latest pip package, or you can install directly from git via pip.

To install the latest pip package:

```
apt-get install python3-pip
pip3 install faucet
```

To install the latest code from git, via pip:

```
pip3 install git+https://github.com/faucetsdn/faucet.git
```

You can then start FAUCET manually:

```
ryu-manager faucet.faucet --verbose
```

Or, you can configure systemd to start the containers automatically:

```
$EDITOR /etc/systemd/system/faucet.service
$EDITOR /etc/systemd/system/gauge.service
systemctl daemon-reload
systemctl enable faucet.service
systemctl enable gauge.service
systemctl restart faucet
systemctl restart gauge
```

`/etc/systemd/system/faucet.service` should contain:

Listing 1.1: faucet.service

```
[Unit]
Description="Faucet OpenFlow switch controller"
After=network-online.target
Wants=network-online.target

[Service]
EnvironmentFile=/etc/default/faucet
User=faucet
Group=faucet
ExecStart=/usr/local/bin/ryu-manager --config-file=${FAUCET_RYU_CONF} --ofp-tcp-
↪listen-port=${FAUCET_LISTEN_PORT} faucet.faucet
ExecReload=/bin/kill -HUP $MAINPID
Restart=always

[Install]
WantedBy=multi-user.target
```

`/etc/systemd/system/gauge.service` should contain:

Listing 1.2: gauge.service

```
[Unit]
Description="Gauge OpenFlow statistics controller"
After=network-online.target
Wants=network-online.target

[Service]
EnvironmentFile=/etc/default/gauge
User=faucet
Group=faucet
ExecStart=/usr/local/bin/ryu-manager --config-file=${GAUGE_RYU_CONF} --ofp-tcp-listen-
↪port=${GAUGE_LISTEN_PORT} --wsapi-host=${WSAPI_LISTEN_HOST} faucet.gauge ryu.app.
↪ofctl_rest
Restart=always

[Install]
WantedBy=multi-user.target
```

1.3 Docker

1.3.1 Faucet Dockerfile

This directory contains three docker files: `Dockerfile`, `Dockerfile.gauge` and `Dockerfile.tests`

1.3.2 Initial configuration

```
sudo mkdir -p /etc/ryu/faucet
sudo vi /etc/ryu/faucet/faucet.yaml
sudo vi /etc/ryu/faucet/gauge.yaml
```

See *Installation* and *Configuration* for configuration options.

In particular, see vendor specific docs for additional files that may be necessary in /etc/ryu/faucet to configure the switch pipeline.

1.3.3 Official builds

We provide official automated builds on Docker Hub so that you can run Faucet easily without having to build your own.

We use Docker tags to differentiate between versions of Faucet. The latest tag will always point to the latest git commit. All tagged versions of Faucet in git are also available to use, for example using the faucet/faucet:v1_3 Docker will run the stable version 1.3 of Faucet.

To pull and run the latest git version of Faucet:

```
mkdir -p /var/log/ryu/faucet/
docker pull faucet/faucet:latest
docker run -d \
    --name faucet \
    -v /etc/ryu/faucet/:/etc/ryu/faucet/ \
```

```
    -v /var/log/ryu/faucet/:/var/log/ryu/faucet/ \
    -p 6653:6653 \
    -p 9302:9302 \
    faucet/faucet
```

Port 6653 is used for OpenFlow, port 9302 is used for Prometheus - port 9302 may be omitted if you do not need Prometheus.

To pull and run the latest git version of Gauge:

```
mkdir -p /var/log/ryu/gauge/
docker pull faucet/gauge:latest
docker run -d \
    --name gauge \
    -v /etc/ryu/faucet/:/etc/ryu/faucet/ \
    -v /var/log/ryu/gauge/:/var/log/ryu/faucet/ \
    -p 6654:6653 \
    -p 9303:9303 \
    faucet/gauge
```

Port 6654 is used for OpenFlow, port 9303 is used for Prometheus - port 9303 may be omitted if you do not need Prometheus.

1.3.4 Dockerfile

All that is needed to run faucet.

It can be built as following:

```
docker build -t faucet/faucet .
```

It can be run as following:

```
mkdir -p /var/log/ryu/faucet/
docker run -d \
    --name faucet \
    -v /etc/ryu/faucet/:/etc/ryu/faucet/ \
    -v /var/log/ryu/faucet/:/var/log/ryu/faucet/ \
    -p 6653:6653 \
    faucet/faucet
```

By default it listens on port 6653 for an OpenFlow switch to connect. Faucet expects to find the configuration file faucet.yaml in the config folder. If needed the -e option can be used to specify the names of files with the FAUCET_LOG, FAUCET_EXCEPTION_LOG, FAUCET_CONFIG environment variables.

1.3.5 Dockerfile.gauge

Runs Gauge.

It can be built as following:

```
docker build -t faucet/gauge -f Dockerfile.gauge .
```

It can be run as following:

```
mkdir -p /var/log/ryu/gauge
docker run -d \
    --name gauge \
    -v /etc/ryu/faucet/:/etc/ryu/faucet/ \
    -v /var/log/ryu/gauge/:/var/log/ryu/gauge/ \
    -p 6654:6653 \
    faucet/gauge
```

By default listens on port 6653. If you are running this with Faucet you will need to modify the port one of the containers listens on and configure your switches to talk to both. The faucet configuration file faucet.yaml should be placed in the config directory, this also should include to configuration for gauge.

1.3.6 docker-compose.yaml

This is an example docker-compose file that can be used to set up gauge to talk to prometheus and influxdb with a grafana instance for dashboards and visualisations.

It can be run with `docker-compose up`

The time-series databases with the default settings will write to `/opt/prometheus/ /opt/influxdb/ shared/data/db` you can edit these locations by modifying the `docker-compose.yaml` file.

On OSX, some of the default shared paths are not accessible, so to overwrite the location that volumes are written to on your host, export an environment varible name `FAUCET_PREFIX` and it will get prepended to the host paths. For example:

```
export FAUCET_PREFIX=/opt/faucet
```

When all the docker containers are running we will need to configure grafana to talk to prometheus and influxdb. First login to the grafana web interface on port 3000 (e.g http://localhost:3000) using the default credentials of `admin:admin`.

Then add two data sources. Use the following settings for prometheus:

```
Name: Prometheus
Type: Prometheus
Url: http://prometheus:9090
Access: proxy
```

And the following settings for influxdb:

```
Name: InfluxDB
Type: InfluxDB
Url: http://influxdb:8086
Access: proxy
With Credentials: true
Database: faucet
User: faucet
Password: faucet
```

Check the connection using test connection.

From here you can add a new dashboard and a graphs for pulling data from the data sources. See the Grafana's documentation for more on how to do this.

1.4 Configuration

Faucet is configured with a YAML-based configuration file, `faucet.yaml`. The following is example demonstrating a few common features:

Listing 1.3: faucet.yaml

```
include:
    - acls.yaml

vlans:
    office:
        vid: 100
        description: "office network"
        acl_in: office-vlan-protect
        faucet_mac: "0e:00:00:00:10:01"
        faucet_vips: ['10.0.100.254/24', '2001:100::1/64', 'fe80::c00:00ff:fe00:1001/
↪64']
        routes:
            - route:
                ip_dst: '192.168.0.0/24'
                ip_gw: '10.0.100.2'
    guest:
        vid: 200
        description: "guest network"
        faucet_mac: "0e:00:00:00:20:01"
        faucet_vips: ['10.0.200.254/24', '2001:200::1/64', 'fe80::c00:00ff:fe00:2001/
↪64']

routers:
    router-office-guest:
        vlans: [office, guest]

dps:
    sw1:
        dp_id: 0x1
        hardware: "Open vSwitch"
        proactive_learn: True
        interfaces:
            1:
                name: "h1"
                description: "host1 container"
                native_vlan: office
                acl_in: access-port-protect
            2:
                name: "h2"
                description: "host2 container"
                native_vlan: office
                acl_in: access-port-protect
            3:
                name: "g1"
                description: "guest1 container"
                native_vlan: guest
                acl_in: access-port-protect
            4:
                name: "s1"
                description: "services1 container"
                native_vlan: office
```

```
                acl_in: service-port-protect
            5:
                name: "trunk"
                description: "VLAN trunk to sw2"
                tagged_vlans: [office]
                acl_in: access-port-protect
    sw2:
        dp_id: 0x2
        hardware: "Allied-Telesis"
        interfaces:
            1:
                name: "pi"
                description: "Raspberry Pi"
                native_vlan: office
                acl_in: access-port-protect
            2:
                name: "laptop"
                description: "Guest Laptop"
                native_vlan: guest
                acl_in: access-port-protect
            24:
                name: "trunk"
                description: "VLAN trunk to sw1"
                tagged_vlans: [office, guest]
```

The datapath ID may be specified as an integer or hex string (beginning with 0x).

A port not explicitly defined in the YAML configuration file will be left down and will drop all packets.

Gauge is configured similarly with, `gauge.yaml`. The following is example demonstrating a few common features:

Listing 1.4: gauge.yaml

```
faucet_configs:
    - '/etc/ryu/faucet/faucet.yaml'
watchers:
    port_status_poller:
        type: 'port_state'
        dps: ['sw1', 'sw2']
        db: 'influx'
    port_stats_poller:
        type: 'port_stats'
        #dps: ['sw1', 'sw2']
        all_dps: True
        interval: 10
        #db: 'influx'
        db: 'prometheus'
    flow_table_poller:
        type: 'flow_table'
        interval: 60
        dps: ['sw1', 'sw2']
        #db: 'couchdb'
        db: 'influx'
dbs:
    ft_file:
        type: 'text'
        compress: True
        file: 'flow_table.yaml.gz'
    prometheus:
```

```
        type: 'prometheus'
        prometheus_addr: '0.0.0.0'
        prometheus_port: 9303
    influx:
        type: 'influx'
        influx_db: 'faucet'
        influx_host: 'influxdb'
        influx_port: 8086
        influx_user: 'faucet'
        influx_pwd: 'faucet'
        influx_timeout: 10
    couchdb:
        type: gaugedb
        gdb_type: nosql
        nosql_db: couch
        db_username: couch
        db_password: 123
        db_ip: 'couchdb'
        db_port: 5984
        driver: 'couchdb'
        views:
            switch_view: '_design/switches/_view/switch'
            match_view: '_design/flows/_view/match'
            tag_view: '_design/tags/_view/tags'
        switches_doc: 'switches_bak'
        flows_doc: 'flows_bak'
        db_update_counter: 2
```

1.4.1 Verifying configuration

You can verify that your configuration is correct with the `check_faucet_config` script:

```
check_faucet_config /etc/ryu/faucet/faucet.yaml
```

1.4.2 Configuration examples

For complete working examples of configuration features, see the unit tests, `tests/faucet_mininet_test.py`. For example, `FaucetUntaggedACLTest` shows how to configure an ACL to block a TCP port, `FaucetTaggedIPv4RouteTest` shows how to configure static IPv4 routing.

1.4.3 Applying configuration updates

You can update FAUCET's configuration by sending it a HUP signal. This will cause it to apply the minimum number of flow changes to the switch(es), to implement the change.

```
pkill -HUP -f faucet.faucet
```

1.4.4 Configuration in separate files

Extra DP, VLAN or ACL data can also be separated into different files and included into the main configuration file, as shown below. The `include` field is used for configuration files which are required to be loaded, and Faucet will

log an error if there was a problem while loading a file. Files listed on `include-optional` will simply be skipped and a warning will be logged instead.

Files are parsed in order, and both absolute and relative (to the configuration file) paths are allowed. DPs, VLANs or ACLs defined in subsequent files overwrite previously defined ones with the same name.

`faucet.yaml`

```
include:
    - /etc/ryu/faucet/dps.yaml
    - /etc/ryu/faucet/vlans.yaml

include-optional:
    - acls.yaml
```

`dps.yaml`

```
# Recursive include is allowed, if needed.
# Again, relative paths are relative to this configuration file.
include-optional:
    - override.yaml

dps:
    test-switch-1:
        ...
    test-switch-2:
        ...
```

1.4.5 Configuration options

DP

Attribute	Default
arp_neighbor_timeout	500
cookie	1524372928
description	None
dp_id	None
drop_bpdu	True
drop_broadcast_source_address	True
drop_lldp	True
drop_spoofed_faucet_mac	True
eth_dst_table	None
eth_src_table	None
flood_table	None
group_table	False
hardware	Open vSwitch
high_priority	None
highest_priority	None
ignore_learn_ins	3
interfaces	{}
ipv4_fib_table	None
ipv6_fib_table	None
learn_ban_timeout	10

Table 1.1 – continu

Attribute	Default
learn_jitter	10
low_priority	None
lowest_priority	None
max_host_fib_retry_count	10
max_hosts_per_resolve_cycle	5
max_resolve_backoff_time	32
name	None
ofchannel_log	None
packetin_pps	0
port_acl_table	None
priority_offset	0
stack	None
table_offset	0
timeout	300
vlan_acl_table	None
vlan_table | None |	

Port

Attribute	Default	Description
acl_in	None	
description	None	
enabled	True	
max_hosts	255	Maximum number of hosts
mirror	None	
mirror_destination	False	
name	None	
native_vlan	None	
number	None	
permanent_learn	False	
stack	None	
tagged_vlans	None	
unicast_flood	True	

Router

Attribute	Default	Description
vlans	None	

VLAN

Attribute	Default	Description
acl_in	None	
bgp_as	0	
bgp_local_address	None	
bgp_neighbor_addresses	[]	
bgp_neighbor_as	None	
bgp_neighbour_addresses	[]	
bgp_neighbour_as	0	
bgp_port	9179	
bgp_routerid		
description	None	
faucet_vips	None	
max_hosts	255	Limit number of hosts that can be learned on a VLAN
name	None	
proactive_arp_limit	None	Don't proactively ARP for hosts if over this limit (None unlimited)
proactive_nd_limit	None	Don't proactively ND for hosts if over this limit (None unlimited)
routes	None	
unicast_flood	True	
vid	None	

1.5 Configuration Recipe Book

In this section we will cover some common network configurations and how you would configure these with the Faucet YAML configuration format.

1.5.1 Forwarding

1.5.2 Routing

1.5.3 Policy

1.6 Vendor-specific Documentation

1.6.1 Faucet on Allied Telesis products

Introduction

Allied Telesis has a wide portfolio of OpenFlow enabled switches that all support the Faucet pipeline. These OpenFlow enabled switches come in various port configurations of 10/18/28/52 with POE+ models as well. Here is a list of some of our most popular switches:

- AT-x930
- AT-x510
- AT-x230

Setup

Switch

OpenFlow supported Firmware

OpenFlow has been supported since AlliedWarePlus version 5.4.6 onwards. To inquire more about compatibility of versions, you can contact our customer support team here.

OpenFlow configuration

For a **Pure OpenFlow** deployment, we recommend the following configurations on the switch. Most of these configuration steps will be shown with an example.

```
/* Create an OpenFlow native VLAN */
awplus (config)# vlan database
awplus (config-vlan)# vlan 4090

/* Set an IP address for Control Plane(CP)
 * Here we will use vlan1 for Management/Control Plane */
awplus (config)# interface vlan1
awplus (config-if)# ip address 192.168.1.1/24

/* Configure the FAUCET controller
 * Let's use TCP port 6653 for connection to Faucet */
awplus (config)# openflow controller tcp 192.168.1.10 6653

/* (OPTIONAL) Configure GAUGE controller
 * Let's use TCP port 6654 for connection to Gauge */
awplus (config)# openflow controller tcp 192.168.1.10 6654

/* User must set a dedicated native VLAN for OpenFlow ports
 * OpenFlow native VLAN MUST be created before it is set!
 * VLAN ID for this native VLAN must be different from the native VLAN for control␣
↪plane */
awplus (config)# openflow native vlan 4090

/* Enable OpenFlow on desired ports */
awplus (config)# interface port1.0.1-1.0.46
awplus (config-if)# openflow

/* Disable Spanning Tree Globally */
awplus (config)# no spanning-tree rstp enable

/* OpenFlow requires that ports under its control do not send any control traffic
 * So it is better to disable RSTP and IGMP Snooping TCN Query Solicitation.
 * Disable IGMP Snooping TCN Query Solicitation on the OpenFlow native VLAN */
awplus (config)# interface vlan4090
awplus (config-if)# no ip igmp snooping tcn query solicit
```

Once OpenFlow is up and running and connected to Faucet/Gauge controller, you should be able to verify the operation using some of our show commands.

```
/* To check contents of the DP flows */
awplus# show openflow flows

/* To check the actual rules as pushed by the controller */
awplus# show openflow rules
```

```
/* To check the OpenFlow configuration and other parameters */
awplus# show openflow status
awplus# show openflow config
awplus# show openflow coverage
```

Some other OPTIONAL configuration commands, that may be useful to modify some parameters, if needed.

```
/* Set the OpenFlow version other than default version(v1.3) */
awplus (config)# openflow version 1.0

/* Set IPv6 hardware filter size
 * User needs to configure the following command if a packet needs to be forwarded by␣
↪IPv6 address matching!
 * Please note that this command is supported on AT-x510 and AT-x930 only */
awplus (config)# platform hwfilter-size ipv4-full-ipv6

/* Set the datapath ID(DPID)
 * By default, we use the switch MAC address for datapath-ID.
 * To change the DPID to a hex value 0x1, use the following */
awplus (config)# openflow datapath-id 1

/* NOTE - For all software versions prior to 5.4.7, all VLAN(s) must be included in␣
↪the vlan database config
 * on the switch before they can be used by OpenFlow.
 * Here is an example to create DP VLANs 2-100 */
awplus (config)# vlan database
awplus (config-vlan)# vlan 2-100
```

Faucet

Edit the faucet configuration file (/etc/ryu/faucet/faucet.yaml) to add the datapath of the switch you wish to be managed by faucet. This yaml file also contains the interfaces that need to be seen by Faucet as openflow ports. The device type (hardware) should be set to **Allied-Telesis** in the configuration file.

```
dps:
    allied-telesis:
        dp_id: 0x0000eccd6d123456
        hardware: "Allied-Telesis"
        interfaces:
            1:
                native_vlan: 100
                name: "port1.0.1"
            2:
                tagged_vlans: [2001,2002,2003]
                name: "port1.0.2"
                description: "windscale"
```

References

- Allied Telesis x930
- OpenFlow Configuration Guide

1.6.2 Faucet on HPE-Aruba Switches

Introduction

All the Aruba's v3 generation of wired switches support the FAUCET pipeline. These switches include:

- 5400R
- 3810
- 2930F

The FAUCET pipeline is only supported from **16.03** release of the firmware onwards.

For any queries, please post your question on HPE's SDN forum.

Setup

Switch

VLAN/PORT configuration

To ensure any port/vlan configuration specified in the *faucet.yaml* file works, one needs to pre-configure all vlans on the switch. Every dataplane port on the switch is made a tagged member of every vlan. This permits FAUCET to perform flow matching and packet-out on any port/vlan combination. The control-plane port (either OOBM or a front-panel port) is kept separate, so that FAUCET does not attempt to modify the control-plane port state.

- Using OOBM control-plane (3810, 5400R)

```
// Increase the maximum number of allowed VLANs on the box and save the configuration.
switch (config)# max-vlans 4094
switch (config)# write mem

// Reboot the box for the new max-vlan configuration to take affect.
switch (config)# boot system

// Configure the control-plane IP address
switch (config)# oobm ip address 20.0.0.1/24

// Create maximum number of VLANs and tag every dataplane port available to each vlan.
↪ Takes up to 30 minutes.
switch (config)# vlan 2-4094 tagged all
```

- Using VLAN control-plane (2930)

```
// Increase the maximum number of allowed VLANs on the box and save the configuration.
switch (config)# max-vlans 2048
switch (config)# write mem

// Reboot the box for the new max-vlan configuration to take affect.
switch (config)# boot system

// Create a control-plane vlan and add a single control-plane port (port 48)
switch (config)# vlan 2048 untagged 48

// Configure the control-plane IP address
switch (config)# vlan 2048 ip address 20.0.0.1/24

// Create maximum number of VLANs and tag every dataplane port available to each vlan,
```

```
// except for the control-plane vlan (above). Note that the command below assumes it
// is run on a 52-port switch, with port 48 as the control-plane. Takes up to 20␣
↪minutes.
switch (config)# vlan 2-2047 tagged 1-47,49-52
```

OpenFlow configuration

Aruba switches reference a controller by ID, so first configure the controllers which will be used. The controller-interface matches the control-plane configuration above.

- Using OOBM control-plane (3810, 5400R)

```
// Enter OpenFlow context
switch (config)# openflow

// Configure an OpenFlow controller connection for FAUCET over tcp-port 6653
switch(openflow)# controller-id 1 ip 20.0.0.2 port 6653 controller-interface oobm

// Configure an OpenFlow controller connection for Gauge over tcp-port 6654
switch(openflow)# controller-id 2 ip 20.0.0.2 port 6654 controller-interface oobm
```

- Using VLAN control-plane (2930)

```
// Enter OpenFlow context
switch (config)# openflow

// Configure an OpenFlow controller connection for FAUCET over tcp-port 6653
switch(openflow)# controller-id 1 ip 20.0.0.2 port 6653 controller-interface vlan 2048

// Configure an OpenFlow controller connection for Gauge over tcp-port 6654
switch(openflow)# controller-id 2 ip 20.0.0.2 port 6654 controller-interface vlan 2048
```

Aruba switches support two OpenFlow instance types:

- **Aggregate** - Every VLAN on the switch apart from the controller/management VLANs are OpenFlow managed.
- **Virtualization** - A set of VLANs configured as members are OpenFlow managed.

Since FAUCET is designed for a pure OpenFlow environment, we choose the "**aggregate**" instance type.

```
// Enter the OpenFlow instance context
switch(openflow)# instance aggregate

// Associate the controllers to the instance
switch(of-inst-aggregate)# controller-id 1
switch(of-inst-aggregate)# controller-id 2

// Configure the OpenFlow version to be 1.3
switch(of-inst-aggregate)# version 1.3 only

// Configure the pipeline model type of the instance. It is a must to set it to␣
↪custom.
switch(of-inst-aggregate)# pipeline-model custom

// Configure the payload in the packet-ins message to be sent in its original form.
switch(of-inst-aggregate)# packet-in vlan-tagging input-form

// Ensure the switch re-attempts an OpenFlow connection at least once
// every 10 seconds when connection is dropped/inactive.
switch(of-inst-aggregate)# max-backoff-interval 10
```

```
// Allow OpenFlow to override some protocols which are otherwise excluded from␣
↪OpenFlow processing in switch CPU.
switch(of-inst-aggregate)# override-protocol all
WARNING: Overriding the protocol can also potentially lead to control packets
         of the protocol to bypass any of the security policies like ACL(s).
Continue (y/n)? y

// Enable the instance
switch(of-inst-aggregate)# enable
switch(of-inst-aggregate)# exit

// Enable OpenFlow globally
switch(openflow)# enable
switch(openflow)# exit

// Check the OpenFlow instance configuration (includes Datapath ID associated)
switch# show openflow instance aggregate
...

// Easier way to get the Datapath ID associated with the OpenFlow instance
switch# show openflow instance aggregate | include Datapath ID
...
```

At this point, OpenFlow is enabled and running on the switch. If the FAUCET controller is running and has connected to the switch successfully, you should see the FAUCET pipeline programmed on the switch.

```
switch# show openflow instance aggregate flow-table

 OpenFlow Instance Flow Table Information

 Table                        Flow     Miss
 ID    Table Name             Count    Count          Goto Table
 ----- ---------------------- -------- -------------- -------------
 0     Port ACL               5        0              1, 2, 3, 4...
 1     VLAN                   10       0              2, 3, 4, 5...
 2     VLAN ACL               1        0              3, 4, 5, 6...
 3     Ethernet Source        2        0              4, 5, 6, 7, 8
 4     IPv4 FIB               1        0              5, 6, 7, 8
 5     IPv6 FIB               1        0              6, 7, 8
 6     VIP                    1        0              7, 8
 7     Ethernet Destination   2        0              8
 8     Flood                  21       0              *


 Table
 ID    Table Name             Available Free Flow Count
 ----- ---------------------- ------------------------------
 0     Port ACL               Ports 1-52          : 46
 1     VLAN                   Ports 1-52          : 91
 2     VLAN ACL               Ports 1-52          : 50
 3     Ethernet Source        Ports 1-52          : 99
 4     IPv4 FIB               Ports 1-52          : 100
 5     IPv6 FIB               Ports 1-52          : 100
 6     VIP                    Ports 1-52          : 20
 7     Ethernet Destination   Ports 1-52          : 99
 8     Flood                  Ports 1-52          : 280
```

```
* Denotes that the pipeline could end here.
```

Faucet

On the FAUCET configuration file (*/etc/ryu/faucet/faucet.yaml*), add the datapath of the switch you wish to be managed by FAUCET. The device type (hardware) should be set to **Aruba** in the configuration file.

```
dps:
    aruba-3810:
        dp_id: 0x00013863bbc41800
        hardware: "Aruba"
        interfaces:
            1:
                native_vlan: 100
                name: "port1"
            2:
                native_vlan: 100
                name: "port2"
```

You will also need to install pipeline configuration files (these files instruct FAUCET to configure the switch with the right OpenFlow tables - these files and FAUCET's pipeline must match).

```
sudo cp etc/ryu/faucet/ofproto_to_ryu.json /etc/ryu/faucet
sudo cp etc/ryu/faucet/aruba_pipeline.json /etc/ryu/faucet
```

Scale

Most tables in the current FAUCET pipeline need wildcards and hence use TCAMs in hardware. There are 2000 entries available globally for the whole pipeline. Currently, it has been distributed amongst the 9 tables as follows:

Table	Maximum Entries
Port ACL	50
VLAN	300
VLAN ACL	50
ETH_SRC	500
IPv4 FIB	300
IPv6 FIB	10
VIP	10
ETH_DST	500
FLOOD	300

Based on one's deployment needs, these numbers can be updated for each table (update max_entries in $(REPO_ROOT)/faucet/aruba/aruba_pipeline.json).

```
NOTE: The summation of max entries across all 9 tables cannot cross 2000 and the␣
↪minimum size of a given table has to be 2.
You need to restart FAUCET for the new numbers to reflect on the switch.
```

Limitations

- Aruba switches currently does not support all the IPv6 related functionality inside FAUCET

- Aruba switches currently does not support the OFPAT_DEC_NW_TTL action (so when routing, TTL will not be decremented).

Debug

If you encounter a failure or unexpected behavior, it may help to enable debug output on Aruba switches. Debug output displays information about what OpenFlow is doing on the switch at message-level granularity.

```
switch# debug openflow
switch# debug destination session
switch# show debug

 Debug Logging

  Source IP Selection: Outgoing Interface
  Origin identifier: Outgoing Interface IP
  Destination:
   Session

  Enabled debug types:
   openflow
   openflow packets
   openflow events
   openflow errors
   openflow packets tx
   openflow packets rx
   openflow packets tx pkt_in
   openflow packets rx pkt_out
   openflow packets rx flow_mod
```

References

- Aruba OpenFlow Administrator Guide (16.03)
- Aruba Switches
- FAUCET

1.6.3 Faucet on Lagopus

Introduction

Lagopus (http://www.lagopus.org/) is a software OpenFlow 1.3 switch, that also supports DPDK.

FAUCET is supported as of Lagopus 0.2.11 (https://github.com/lagopus/lagopus/issues/107).

Setup

Lagopus install on a supported Linux distribution

Install Lagopus (https://github.com/lagopus/lagopus/blob/master/QUICKSTART.md). You don't need to install Ryu since we will be using FAUCET and FAUCET's installation takes care of that dependency.

These instructions are for Ubuntu 16.0.4 (without DPDK). In theory any distribution, with or without DPDK, that Lagopus supports will work with FAUCET.

Create lagopus.dsl configuration file

In this example, Lagopus is controlling two ports, enp1s0f0 and enp1s0f1, which will be known as OpenFlow ports 1 and 2 on DPID 0x1. FAUCET and Lagopus are running on the same host (though of course, they don't need to be).

```
$ cat /usr/local/etc/lagopus/lagopus.dsl
channel channel01 create -dst-addr 127.0.0.1 -protocol tcp

controller controller01 create -channel channel01 -role equal -connection-type main

interface interface01 create -type ethernet-rawsock -device enp1s0f0

interface interface02 create -type ethernet-rawsock -device enp1s0f1

port port01 create -interface interface01

port port02 create -interface interface02

bridge bridge01 create -controller controller01 -port port01 1 -port port02 2 -dpid␣
↪0x1
bridge bridge01 enable
```

Create faucet.yaml

```
$ cat /etc/ryu/faucet/faucet.yaml
vlans:
    100:
        name: "test"
dps:
    lagopus-1:
        dp_id: 0x1
        hardware: "Lagopus"
        interfaces:
            1:
                native_vlan: 100
            2:
                native_vlan: 100
```

Start Lagopus

Start in debug mode, in a dedicated terminal.

```
# lagopus -d
```

Run FAUCET

```
$ ryu-manager --config-file=/home/faucet/faucet/etc/ryu/ryu.conf /home/faucet/faucet/
↪faucet/faucet.py --verbose --ofp-listen-host=127.0.0.1
```

Test connectivity

Host(s) on enp1s0f0 and enp1s0f1 in the same IP subnet, should now be able to communicate, and FAUCET's log file should indicate learning is occurring:

```
$ tail /var/log/ryu/faucet/faucet.log
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) Configuring DP
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) Delete VLAN vid:100 ports:1,2
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) VLANs changed/added: [100]
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) Configuring VLAN vid:100 ports:1,2
```

```
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) Configuring VLAN vid:100 ports:1,2
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) Port 1 added
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) Sending config for port 1
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) Port 2 added
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) Sending config for port 2
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) Packet_in src:00:16:41:6d:87:28 in_
↪port:1 vid:100
May 11 13:04:57 faucet.valve INFO     learned 1 hosts on vlan 100
May 11 13:04:57 faucet.valve INFO     DPID 1 (0x1) Packet_in src:00:16:41:32:87:e0 in_
↪port:2 vid:100
May 11 13:04:57 faucet.valve INFO     learned 2 hosts on vlan 100
```

1.6.4 Faucet on ZodiacFX

Introduction

ZodiacFX is a small 4 port multi table OF1.3 switch from Northbound Networks (https://northboundnetworks.com/products/zodiac-fx).

Caveats

- ZodiacFX allows only one controller (so you cannot run Gauge).
- The default OF port is 6633; it is recommended to use 6653.
- It is recommended to enable ether type filtering to minimize corrupt packets.

Applying recommended config

You can use the following expect script to program the recommended configuration:

Listing 1.5: conf-zodiac.sh

```
#!/usr/bin/expect

##
## configure ZodiacFX with recommended settings.
##

# Serial port assigned to ZodiacFX
set port /dev/ttyACM0

set timeout 5
set prompt {Zodiac_FX\#}
set configprompt {Zodiac_FX\(config\)\#}
set spawned [spawn -open [open $port w+]]

send_user "get initial prompt\n"
send "\r"
send "\r"
expect -re $prompt
send_user "found initial prompt\n"
send "config\r"
expect -re $configprompt
```

```
send_user "setting ethertype-filter\n"
send "set ethertype-filter enable\r"
expect -re $configprompt
send_user "setting of-port"
send "set of-port 6653\r"
expect -re $configprompt
send "save\r"
expect -re $configprompt
send "exit\r"
expect -re $prompt
send "restart\r"
expect -re "Restarting"
```

Example of running the script:

```
# ./conf-zodiac.sh
spawn [open ...]
get initial prompt


 _____                ___                  ________  __
/__   /  _____  _____/ (_)___ _______    / _____/ |/ /
  / /  / __ \/ __  / / __ `/ ___/  / /_   |   /
 / /__/ /_/ / /_/ / / /_/ / /__   / __/  /   |
/_____/\_____/\__,_/_/\__,_/\___/  /_/    /_/|_|
                by Northbound Networks


Type 'help' for a list of available commands

Zodiac_FX#
Zodiac_FX# found initial prompt
config
Zodiac_FX(config)# setting ethertype-filter
set ethertype-filter enable
EtherType Filtering Enabled
Zodiac_FX(config)# setting of-portset of-port 6653
OpenFlow Port set to 6653
Zodiac_FX(config)# save
Writing Configuration to EEPROM (197 bytes)
Zodiac_FX(config)# exit
Zodiac_FX# restart
Restarting the Zodiac FX, please reopen your terminal application.
```

1.6.5 Faucet on NoviFlow

Introduction

NoviFlow provide a range of switches known to work with FAUCET.

These instructions have been tested on NS1248, NS1132, NS2116, NS2128, NS2122, NS2150, NS21100 switches, using software versions NW400.1.8 to NW400.3.1, running with FAUCET v1.6.4.

When using a more recent FAUCET version, different table configurations may be required.

Setup

Configure the CPN on the switch

In this example, the server running FAUCET is 10.0.1.8; configuration for CPN interfaces is not shown.

```
set config controller controllergroup 1 controllerid 1 priority 1 ipaddr 10.0.1.8␣
↪port 6653 security none
set config switch dpid 0x1
```

Configure the tables

These matches are known to pass the unit tests as of FAUCET 1.6.4, but take care to adjust ACL table matches and any changes for future versions.

```
set config pipeline tablesizes 1024 1024 1024 1024 1024 1024 1024 1024 1024␣
↪tablewidths 80 40 40 40 40 40 40 40 40
set config table tableid 0 matchfields 0 3 4 5 6 10 14 23 29 31
set config table tableid 1 matchfields 0 3 4 5 6
set config table tableid 2 matchfields 0 5 6 10 11 12 14
set config table tableid 3 matchfields 0 3 4 5 6 10 29
set config table tableid 4 matchfields 5 6 12
set config table tableid 5 matchfields 5 6 27
set config table tableid 6 matchfields 3 5 10 23
set config table tableid 7 matchfields 0 3 6
set config table tableid 8 matchfields 0 3 6
```

Create faucet.yaml

```
$ cat /etc/ryu/faucet/faucet.yaml
vlans:
    100:
        name: "test"
dps:
    noviflow-1:
        dp_id: 0x1
        hardware: "NoviFlow"
        interfaces:
            1:
                native_vlan: 100
            2:
                native_vlan: 100
```

Run FAUCET

```
$ ryu-manager faucet.faucet --verbose
```

Test connectivity

Host(s) on ports 1 and 2 should now be able to communicate, and FAUCET's log file should indicate learning is occurring:

```
May 14 17:06:15 faucet DEBUG    DPID 1 (0x1) connected
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Configuring DP
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Delete VLAN vid:100 ports:1,2,3,4
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) VLANs changed/added: [100]
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Configuring VLAN vid:100 ports:1,2,
↪3,4
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Configuring VLAN vid:100 ports:1,2,
↪3,4
```

```
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Port 1 added
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Sending config for port 1
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Port 2 added
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Sending config for port 2
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Port 3 added
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Sending config for port 3
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Port 4 added
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Sending config for port 4
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Packet_in src:62:4c:f5:bb:33:3c in_
↪port:2 vid:100
May 14 17:06:15 faucet.valve INFO     learned 1 hosts on vlan 100
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Packet_in src:62:4c:f5:bb:33:3c in_
↪port:2 vid:100
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Packet_in src:2a:e1:65:3c:49:e4 in_
↪port:3 vid:100
May 14 17:06:15 faucet.valve INFO     DPID 1 (0x1) Packet_in src:2a:e1:65:3c:49:e4 in_
↪port:3 vid:100
May 14 17:06:15 faucet.valve INFO     learned 2 hosts on vlan 100
```

1.6.6 Faucet on OVS with DPDK

Introduction

Open vSwitch (http://openvswitch.org/) is a software OpenFlow switch, that supports DPDK. It is also the reference switching platform for FAUCET.

Setup

Install OVS on a supported Linux distribution

Install OVS and DPDK per the OVS instructions, including enabling DPDK at compile time and in OVS's initial configuration (http://docs.openvswitch.org/en/latest/intro/install/dpdk/).

These instructions are known to work for Ubuntu 16.0.4, with OVS 2.7.0 and DPDK 16.11.1, kernel 4.4.0-77. In theory later versions of these components should work without changes. A multiport NIC was used, based on the Intel 82580 chipset.

Bind NIC ports to DPDK

NOTE: if you have a multiport NIC, you must bind all the ports on the NIC to DPDK, even if you do not use them all.

From the DPDK source directory, determine the relationship between the interfaces you want to use with DPDK and their PCI IDs:

```
export DPDK_DIR=`pwd`
$DPDK_DIR/tools/dpdk-devbind.py --status
```

In this example, we want to use enp1s0f0 and enp1s0f1.

```
# ./tools/dpdk-devbind.py --status

Network devices using DPDK-compatible driver
============================================
<none>

Network devices using kernel driver
```

```
==================================
0000:01:00.0 '82580 Gigabit Network Connection' if=enp1s0f0 drv=igb unused=
0000:01:00.1 '82580 Gigabit Network Connection' if=enp1s0f1 drv=igb unused=
0000:01:00.2 '82580 Gigabit Network Connection' if=enp1s0f2 drv=igb unused=
0000:01:00.3 '82580 Gigabit Network Connection' if=enp1s0f3 drv=igb unused=
```

Still from the DPDK source directory:

```
export DPDK_DIR=`pwd`
modprobe vfio-pci
chmod a+x /dev/vfio
chmod 0666 /dev/vfio/*
$DPDK_DIR/tools/dpdk-devbind.py --bind=vfio-pci 0000:01:00.0 0000:01:00.1 0000:01:00.
↪2 0000:01:00.3
$DPDK_DIR/tools/dpdk-devbind.py --status
```

Confirm OVS has been configured to use DPDK

```
# /usr/local/share/openvswitch/scripts/ovs-ctl stop
* Exiting ovs-vswitchd (20510)
* Exiting ovsdb-server (20496)
# /usr/local/share/openvswitch/scripts/ovs-ctl start
* Starting ovsdb-server
* system ID not configured, please use --system-id
* Configuring Open vSwitch system IDs
EAL: Detected 4 lcore(s)
EAL: Probing VFIO support...
EAL: VFIO support initialized
EAL: PCI device 0000:01:00.0 on NUMA socket -1
EAL:   probe driver: 8086:150e net_e1000_igb
EAL:   using IOMMU type 1 (Type 1)
EAL: PCI device 0000:01:00.1 on NUMA socket -1
EAL:   probe driver: 8086:150e net_e1000_igb
EAL: PCI device 0000:01:00.2 on NUMA socket -1
EAL:   probe driver: 8086:150e net_e1000_igb
EAL: PCI device 0000:01:00.3 on NUMA socket -1
EAL:   probe driver: 8086:150e net_e1000_igb
EAL: PCI device 0000:02:00.0 on NUMA socket -1
EAL:   probe driver: 8086:150e net_e1000_igb
EAL: PCI device 0000:02:00.1 on NUMA socket -1
EAL:   probe driver: 8086:150e net_e1000_igb
EAL: PCI device 0000:02:00.2 on NUMA socket -1
EAL:   probe driver: 8086:150e net_e1000_igb
EAL: PCI device 0000:02:00.3 on NUMA socket -1
EAL:   probe driver: 8086:150e net_e1000_igb
Zone 0: name:<rte_eth_dev_data>, phys:0x7ffced40, len:0x30100, virt:0x7f843ffced40,␣
↪socket_id:0, flags:0
* Starting ovs-vswitchd
* Enabling remote OVSDB managers
```

Configure an OVS bridge with the DPDK ports

```
ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev protocols=OpenFlow13
ovs-vsctl add-port br0 dpdk0 -- set interface enp1s0f0 type=dpdk options:dpdk-
↪devargs=0000:01:00.0
ovs-vsctl add-port br0 dpdk1 -- set interface enp1s0f1 type=dpdk options:dpdk-
↪devargs=0000:01:00.1
ovs-vsctl set-fail-mode br0 secure
```

```
ovs-vsctl set-controller br0 tcp:127.0.0.1:6653
ovs-vsctl show br0
ovs-vsctl get bridge br0 datapath_id
```

Create faucet.yaml

NOTE: change dp_id, to the value reported above, prefaced with "0x".

```
$ cat /etc/ryu/faucet/faucet.yaml
vlans:
    100:
        name: "test"
dps:
    ovsdpdk-1:
        dp_id: 0x000090e2ba7e7564
        hardware: "Open vSwitch"
        interfaces:
            1:
                native_vlan: 100
            2:
                native_vlan: 100
```

Run FAUCET

```
$ ryu-manager faucet.faucet --verbose --ofp-listen-host=127.0.0.1
```

Test connectivity

Host(s) on enp1s0f0 and enp1s0f1 in the same IP subnet, should now be able to communicate, and FAUCET's log file should indicate learning is occurring:

```
May 11 14:53:32 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564)␣
↪Configuring DP
May 11 14:53:32 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564) Delete␣
↪VLAN vid:100 ports:1,2
May 11 14:53:32 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564) VLANs␣
↪changed/added: [100]
May 11 14:53:32 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564)␣
↪Configuring VLAN vid:100 ports:1,2
May 11 14:53:32 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564)␣
↪Configuring VLAN vid:100 ports:1,2
May 11 14:53:32 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564) Port 1␣
↪added
May 11 14:53:32 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564) Sending␣
↪config for port 1
May 11 14:53:32 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564) Port 2␣
↪added
May 11 14:53:32 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564) Sending␣
↪config for port 2
May 11 14:53:33 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564) Packet_in␣
↪src:00:16:41:6d:87:28 in_port:1 vid:100
May 11 14:53:33 faucet.valve INFO     learned 1 hosts on vlan 100
May 11 14:53:33 faucet.valve INFO     DPID 159303465858404 (0x90e2ba7e7564) Packet_in␣
↪src:00:16:41:32:87:e0 in_port:2 vid:100
May 11 14:53:33 faucet.valve INFO     learned 2 hosts on vlan 100
```

1.7 External Resources

1.7.1 Online Tutorials

- http://docs.openvswitch.org/en/latest/tutorials/faucet/
- http://costiser.ro/2017/03/07/sdn-lesson-2-introducing-faucet-as-an-openflow-controller/
- https://inside-openflow.com/openflow-tracks/faucet-controller-application-technical-track/

1.7.2 Tutorial Videos

- https://www.youtube.com/watch?v=fuqzzjmcwlI

CHAPTER 2

Developer Documentation

2.1 Developer guide

This file contains an overview of architecture, coding design/practices, testing and style.

2.1.1 Before submitting a PR

- All unit tests must pass (please use the docker based tests; see *Software switch testing with docker*).
- It is strongly recommended to enable TravisCI testing on your repo. This enables the maintainers to quickly verify that your changes pass all tests in a pristine environment.
- You must add a test if FAUCET's functionality changes (ie. a new feature, or correcting a bug).
- pylint must show no new errors or warnings.
- Code must conform to the style guide (see below).

2.1.2 Code style

Please use the coding style documented at http://google.github.io/styleguide/pyguide.html. Existing code not using this style will be incrementally migrated to comply with it. New code should comply.

2.1.3 Makefile

Makefile is provided at the top level of the directory. Output of `make` is normally stored in `dist` directory. The following are the targets that can be used:

- **uml**: Uses `pyreverse` to provide code class diagrams.
- **dot**: Uses `dot` to provide hirearchical representation of `faucet.yaml` based on `docs/images/faucet-yaml.dot` file

- **codefmt**: Provides command line usage to "Code Style" the Python file
- **codeerrors**: Uses `pylint` on all Python files to generate a code error report and is placed in `dist` directory.
- **stats**: Provides a list of all commits since the last release tag.
- **release**: Used for releasing FAUCET to the next version, Requires `version` and `next_version` variables.

To *directly install* faucet from the cloned git repo, you could use `sudo python setup.py install` command from the root of the directory.

To *build pip installable package*, you could use `python setup.py sdist` command from the root of the directory.

To *remove* any temporarily created directories and files, you could use `rm -rf dist *egg-info` command.

2.1.4 Key architectural concepts/assumptions:

FAUCET's architecture depends on key assumptions, which must be kept in mind at all times.

- FAUCET is the only controller for the switch, that can add or remove flows.
- All supported dataplanes must implement OpenFlow functionally (hardware, software or both) identically. No TTP or switch specific drivers.

In addition:

- FAUCET provisions default deny flows (all traffic not explicitly programmed is dropped).
- Use of packet in is minimized.

FAUCET depends upon these assumptions to guarantee that the switch is always in a known and consistent state, which in turn is required to support high availability (FAUCET provides high availability, through multiple FAUCET controllers using the same version of configuration - any FAUCET can give the switch a consistent response - no state sharing between controllers is required). The FAUCET user can program customized flows to be added to the switch using FAUCET ACLs (see below).

FAUCET also programs the dataplane to do flooding (where configured). This minimizes the use of packet in. This is necessary to reduce competition between essential control plane messages (adding and removing flows), and traffic from the dataplane on the limited bandwidth OpenFlow control channel. Unconstrained packet in messages impact the switch CPU, may overwhelm the OpenFlow control channel, and will expose the FAUCET controller to unvalidated dataplane packets, all of which are security and reliability concerns. In future versions, packet in will be eliminated altogether. The FAUCET user is expected to use policy based forwarding (eg ACLs that redirect traffic of interest to high performance dataplane ports for NFV offload), not packet in.

FAUCET requires all supported dataplanes to implement OpenFlow (specifically, a subset of OpenFlow 1.3) in a functionally identical way. This means that there is no switch-specific driver layer - the exact same messages are sent, whether the switch is OVS or hardware. While this does prevent some earlier generation OpenFlow switches from being supported, commercially available current hardware does not have as many restrictions, and eliminating the need for a switch-specific (or TTP) layer greatly reduces implementation complexity and increases controller programmer productivity.

2.2 Architecture

2.2.1 Faucet Design and Architecture

Faucet enables practical SDN for the masses (see http://queue.acm.org/detail.cfm?id=3015763).

- Drop in/replacement for non-SDN L2/L3 IPv4/IPv6 switch/router (easy migration)

- Packet forwarding/flooding/multicasting done entirely by switch hardware (controller only notified on topology change)
- BGP and static routing (other routing protocols provided by NFV)
- Multi vendor/platform support using OpenFlow 1.3 multi table
- Multi switch, vendor neutral "stacking" (Faucet distributed switching, loop free topology without spanning tree)
- ACLs, as well as allow/drop, allow packets to be copied/rewritten for external NFV applications
- Monitored with Prometheus
- Small code base with high code test coverage and automated testing both hardware and software

See unit and integration tests for working configuration examples.

2.2.2 Faucet Openflow Switch Pipeline

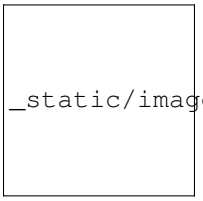


Table 0: PORT_ACL

- Apply user supplied ACLs to a port and send to next table

Table 1: VLAN

- Match fields: `eth_dst, eth_type, in_port, vlan_vid`
- **Operations:**
  - Drop unwanted L2 protocol traffic (and spoofing of Faucet's virtual MAC)
  - **For tagged ports**
    - Match VLAN_VID and send to next table
  - **For untagged ports**
    - Push VLAN frame onto packet with VLAN_VID representing ports native VLAN and send to next table
  - Unknown traffic is dropped

Table 2: VLAN_ACL

- Apply user supplied ACLs to a VLAN and send to next table

Table 3: ETH_SRC

- Match fields: `eth_dst, eth_src, eth_type, in_port, vlan_vid`
- **Operations:**
  - For IPv4/IPv6 traffic where Faucet is the next hop, send to IPV4_FIB or IPV6_FIB (route)
  - For known source MAC, send to ETH_DST (switch)
  - For unknown source MACs, copy header to controller via packet in (for learning) and send to FLOOD

Table 4: IPV4_FIB

- Match fields: `eth_type, ipv4_dst, vlan_vid`
- **Operations:**
  - Route IPv4 traffic to a next-hop for each route we have learned
  - Set eth_src to Faucet's magic MAC address
  - Set eth_dst to the resolved MAC address for the next-hop
  - Decrement TTL
  - Send to ETH_DST table
  - Unknown traffic is dropped

Table 5: IPV6_FIB

- Match fields: `eth_type, ipv6_dst, vlan_vid`
- **Operations:**
  - Route IPv4 traffic to a next-hop for each route we have learned
  - Set eth_src to Faucet's magic MAC address
  - Set eth_dst to the resolved MAC address for the next-hop
  - Decrement TTL
  - Send to ETH_DST table
  - Unknown traffic is dropped

Table 6: VIP

- Match fields: `arp_tpa, eth_dst, eth_type, icmpv6_type, ip_proto`
- **Operations:**
  - Send traffic destined for FAUCET VIPs including IPv4 ARP and IPv6 ND to the controller.
  - IPv6 ND traffic may be flooded also (sent to FLOOD)

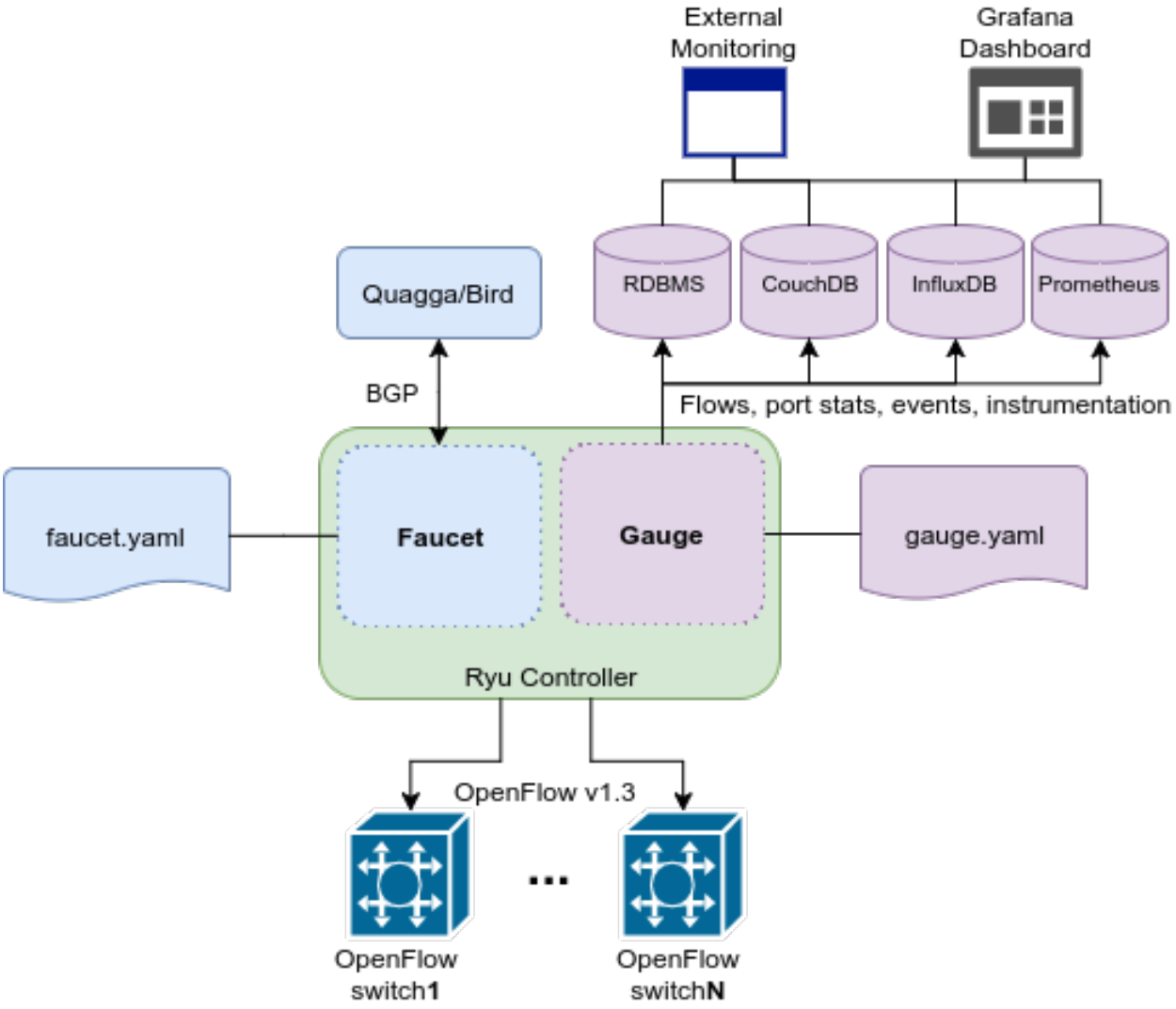
Table 7: ETH_DST

- Match fields: `eth_dst, in_port, vlan_vid`
- **Operations:**
  - For destination MAC addresses we have learned output packet towards that host (popping VLAN frame if we are outputting on an untagged port)
  - Unknown traffic is sent to FLOOD table

Table 8: FLOOD

- Match fields: `eth_dst, in_port, vlan_vid`
- **Operations:**
  - Flood broadcast within VLAN
  - Flood multicast within VLAN
  - Unknown traffic is flooded within VLAN

2.2.3 Faucet Architecture



2.3 Testing

2.3.1 Software switch testing with docker

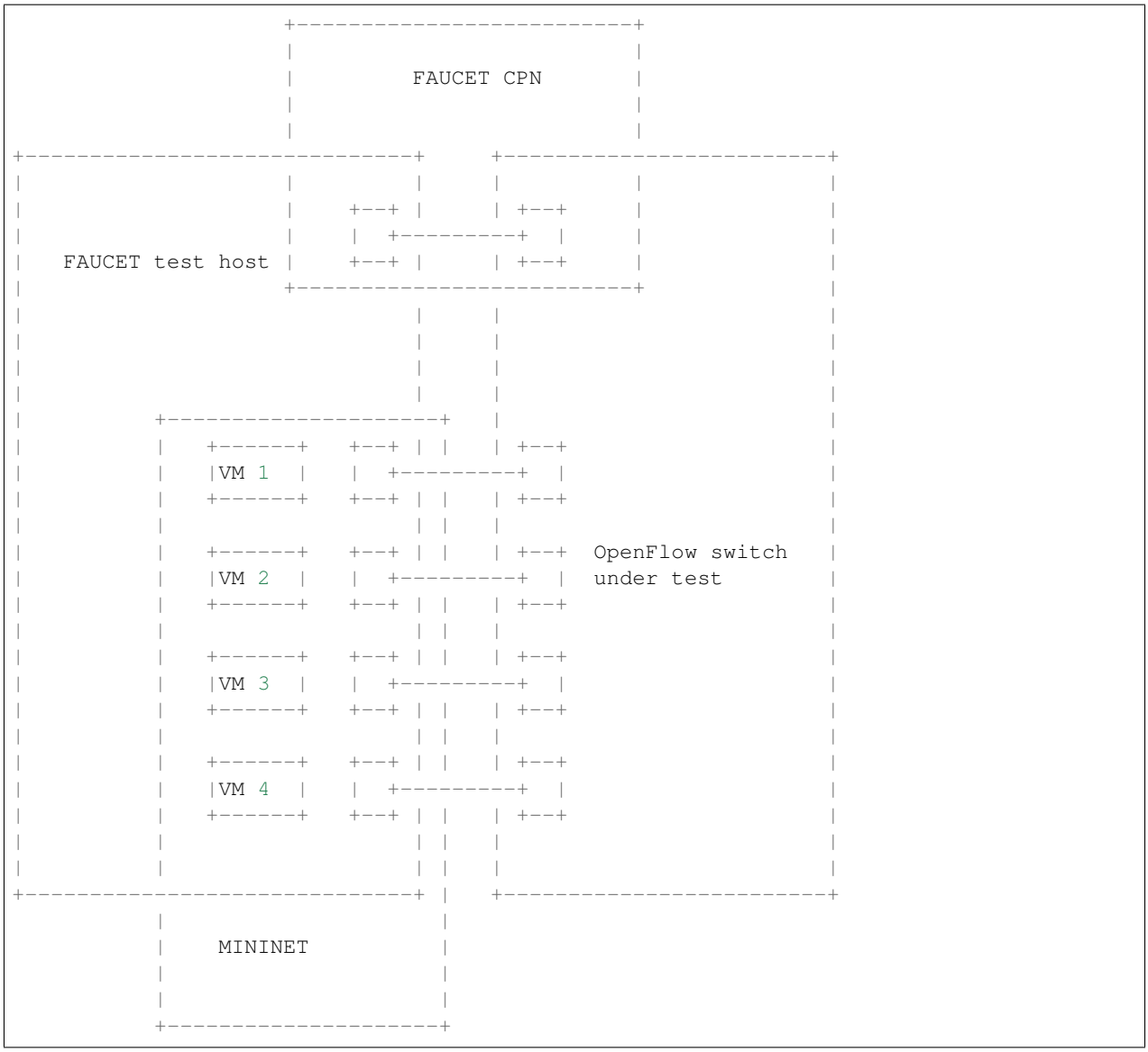
First, get yourself setup with docker based on our *Docker* documentation.

Then you can build and run the mininet tests from the docker entry-point:

```
docker build -t faucet/tests -f Dockerfile.tests .
apparmor_parser -R /etc/apparmor.d/usr.sbin.tcpdump
modprobe openvswitch
sudo docker run --privileged -ti faucet/tests
```

The apparmor command is currently required on Ubuntu hosts to allow the use of tcpdump inside the container.

2.3.2 Hardware switch testing with docker



Requirements

Your test host, requires at least 5 interfaces. 4 interfaces to connect to the dataplane, and one for the CPN for OpenFlow. You will need to assign an IP address to the CPN interface on the host, and configure the switch with a CPN IP address and establish that they can reach each other (eg via ping).

You will need to configure the switch with two OpenFlow controllers, both with the host's CPN IP address, but with different ports (defaults are given below for *of_port* and *gauge_of_port*).

It is assumed that you execute all following commands from your FAUCET source code directory (eg one you have git cloned).

Test configuration

Create a directory for the test configuration:

```
mkdir -p /etc/ryu/faucet
$EDITOR /etc/ryu/faucet/hw_switch_config.yaml
```

`hw_switch_config.yaml` should contain the correct configuration for your switch:

```
hw_switch: True
hardware: 'Open vSwitch'
# Map ports on the hardware switch, to physical ports on this machine.
# If using a switch with less than 4 dataplane ports available, run
# FaucetZodiac tests only. A 4th port must still be defined here and
# must exist, but will not be used.
dp_ports:
  1: enp1s0f0
  2: enp1s0f1
  3: enp1s0f2
  4: enp1s0f3
# Hardware switch's DPID
dpid: 0xeccd6d9936ed
# Port on this machine that connects to hardware switch's CPN port.
# Hardware switch must use IP address of this port as controller IP.
cpn_intf: enp5s0
# There must be two controllers configured on the hardware switch,
# with same IP (see cpn_intf), but different ports - one for FAUCET,
# one for Gauge.
of_port: 6636
gauge_of_port: 6637
# If you wish to test OF over TLS to the hardware switch,
# set the following parameters per Ryu documentation.
# https://github.com/osrg/ryu/blob/master/doc/source/tls.rst
# ctl_privkey: ctl-privkey.pem
# ctl_cert: ctl-cert.pem
# ca_certs: /usr/local/var/lib/openvswitch/pki/switchca/cacert.pem
```

Running the tests

```
docker build -t faucet/tests -f Dockerfile.tests .
apparmor_parser -R /etc/apparmor.d/usr.sbin.tcpdump
modprobe openvswitch
sudo docker run --privileged --net=host \
    -v /etc/ryu/faucet:/etc/ryu/faucet \
    -v /tmp:/tmp \
    -ti faucet/tests
```

Running a single test

```
sudo docker run --privileged --net=host \
    -e FAUCET_TESTS="FaucetUntaggedTest" \
    -v /etc/ryu/faucet:/etc/ryu/faucet \
    -v /tmp:/tmp \
    -ti faucet/tests
```

Checking test results

If a test fails, you can look in /tmp - there will be subdirectories created for each test, which will contain all the logs and debug information (including tcpdumps).

2.4 Fuzzing

2.4.1 Fuzzing faucet with docker

First, get yourself setup with docker based on our *Docker* documentation.

Then you can build and run the afl-fuzz tests:

```
docker build -t faucet/fuzzer -f dockerfile.fuzz .

docker run -d \
  -u $(id -u $USER) \
  --name fuzzer \
  -v /var/log/afl/:/var/log/afl/ \
  faucet/fuzzer
```

AFL then will run indefinitely. You can find the output in /var/log/afl/.

2.5 Source Code

2.5.1 faucet

faucet package

Submodules

faucet.acl module

Configuration for ACLs.

class faucet.acl.**ACL**(*_id*, *dp_id*, *conf*)

Bases: *faucet.conf.Conf*

Implement FAUCET configuration for an ACL.

defaults = {'rules': None, 'exact_match': False}

defaults_types = {'rules': <class 'list'>, 'exact_match': <class 'bool'>}

exact_match = None

mirror_destinations = set()

rules = None

to_conf()

faucet.check_faucet_config module

Standalone script to check FAUCET configuration, return 0 if provided config OK.

faucet.check_faucet_config.**check_config**(*conf_files*)

faucet.check_faucet_config.**main**()

faucet.conf module

Base configuration implementation.

class faucet.conf.**Conf**(*_id*, *dp_id*, *conf=None*)
Bases: object

Base class for FAUCET configuration.

check_config()
As far as possible, check config at instantiation time for errors, typically via assert.

conf_hash(*dyn=False*, *subconf=True*, *ignore_keys=None*)

defaults = {}

defaults_types = {}

dyn_finalized = False

dyn_hash = None

finalize()
Configuration parsing marked complete.

ignore_subconf(*other*, *ignore_keys=None*)
Return True if this config same as other, ignoring sub config.

merge_dyn(*other_conf*)
Merge dynamic state from other conf object.

set_defaults()
Set default values and run any basic sanity checks.

to_conf()
Return configuration as a dict.

update(*conf*)
Parse supplied YAML config and sanity check.

exception faucet.conf.**InvalidConfigError**
Bases: Exception

This error is thrown when the config file is not valid.

faucet.config_parser module

Implement configuration file parsing.

faucet.config_parser.**dp_parser**(*config_file*, *logname*)

faucet.config_parser.**get_config_for_api**(*valves*)
Return config as dict for all DPs.

faucet.config_parser.**watcher_parser**(*config_file*, *logname*, *prom_client*)
Return Watcher instances from config.

faucet.config_parser_util module

Utility functions supporting FAUCET/Gauge config parsing.

faucet.config_parser_util.**config_changed**(*top_config_file*, *new_top_config_file*, *config_hashes*)
Return True if configuration has changed.

Args: top_config_file (str): name of FAUCET config file new_top_config_file (str): name, possibly new, of FAUCET config file. config_hashes (dict): map of config file/includes and hashes of contents.

Returns: bool: True if the file, or any file it includes, has changed.

faucet.config_parser_util.**config_file_hash**(*config_file_name*)
Return hash of YAML config file contents.

faucet.config_parser_util.**dp_config_path**(*config_file*, *parent_file=None*)
Return full path to config file.

faucet.config_parser_util.**dp_include**(*config_hashes*, *config_file*, *logname*, *top_confs*)

faucet.config_parser_util.**get_logger**(*logname*)
Return logger instance for config parsing.

faucet.config_parser_util.**read_config**(*config_file*, *logname*)
Return a parsed YAML config file or None.

faucet.dp module

Configuration for a datapath.

class faucet.dp.**DP**(*_id*, *dp_id*, *conf*)
Bases: *faucet.conf.Conf*

Implement FAUCET configuration for a datapath.

acls = None

add_acl(*acl_ident*, *acl*)
Add an ACL to this DP.

add_port(*port*)
Add a port to this DP.

add_router(*router_ident*, *router*)
Add a router to this DP.

add_vlan(*vlan*)
Add a VLAN to this datapath.

advertise_interval = None

all_valve_tables()
Return list of all Valve tables.

arp_neighbor_timeout = None

check_config()

configured = False

cookie = None

defaults = {'drop_bpdu': True, 'interfaces': {}, 'priority_offset': 0, 'drop_lldp':

defaults_types = {'drop_bpdu': <class 'bool'>, 'interfaces': <class 'dict'>, 'priori

dp_id = None

drop_bpdu = None

drop_broadcast_source_address = None

drop_lldp = None

drop_spoofed_faucet_mac = None

finalize_config(*dps*)
Perform consistency checks after initial config parsing.

get_config_changes(*logger*, *new_dp*)
Detect any config changes.

Args: logger (ValveLogger): logger instance new_dp (DP): new dataplane configuration.

Returns:

changes (tuple) of: deleted_ports (set): deleted port numbers. changed_ports (set): changed/added port numbers. changed_acl_ports (set): changed ACL only port numbers. deleted_vlans (set): deleted VLAN IDs. changed_vlans (set): changed/added VLAN IDs. all_ports_changed (bool): True if all ports changed.

get_config_dict()
Return DP config as a dict for API call.

get_native_vlan(*port_num*)
Return native VLAN for a port by number, or None.

get_tables()
Return tables as dict for API call.

group_table = False

group_table_routing = False

groups = None

high_priority = None

ignore_learn_ins = None

in_port_tables()
Return list of tables that specify in_port as a match.

interface_ranges = None

interfaces = None

learn_ban_timeout = None

learn_jitter = None

low_priority = None

match_tables(*match_type*)
Return list of tables with matches of a specific match type.

max_host_fib_retry_count = None

max_hosts_per_resolve_cycle = None

max_resolve_backoff_time = None

meters = {}

name = None

packetin_pps = None

peer_stack_up_ports(*peer_dp*)
Return list of stack ports that are up towards a peer.

pipeline_config_dir = None

ports = None

priority_offset = None

proactive_learn = None

resolve_stack_topology(*dps*)
Resolve inter-DP config for stacking.

routers = None

running = False

set_defaults()

shortest_path(*dest_dp*)
Return shortest path to a DP, as a list of DPs.

shortest_path_port(*dest_dp*)
Return first port on our DP, that is the shortest path towards dest DP.

shortest_path_to_root()
Return shortest path to root DP, as list of DPs.

stack = None

stack_ports = None

tables = {}

tables_by_id = {}

timeout = None

to_conf()
Return DP config as dict.

use_idle_timeout = None

vlan_match_tables()
Return list of tables that specify vlan_vid as a match.

vlans = None

wildcard_table = <faucet.valve_table.ValveTable object>

faucet.faucet module

RyuApp shim between Ryu and Valve.

class faucet.faucet.**EventFaucetAdvertise**
Bases: ryu.controller.event.EventBase

Event used to trigger periodic network advertisements (eg IPv6 RAs).

class faucet.faucet.**EventFaucetExperimentalAPIRegistered**
Bases: ryu.controller.event.EventBase

Event used to notify that the API is registered with Faucet.

class faucet.faucet.**EventFaucetMetricUpdate**
Bases: ryu.controller.event.EventBase

Event used to trigger update of metrics.

class faucet.faucet.**EventFaucetReconfigure**
Bases: ryu.controller.event.EventBase

Event used to trigger FAUCET reconfiguration.

class faucet.faucet.**EventFaucetResolveGateways**
Bases: ryu.controller.event.EventBase

Event used to trigger gateway re/resolution.

class faucet.faucet.**EventFaucetStateExpire**
Bases: ryu.controller.event.EventBase

Event used to trigger expiration of state in controller.

class faucet.faucet.**Faucet**(**args*, ***kwargs*)
Bases: ryu.base.app_manager.RyuApp

A RyuApp that implements an L2/L3 learning VLAN switch.

Valve provides the switch implementation; this is a shim for the Ryu event handling framework to interface with Valve.

OFP_VERSIONS = [4]

advertise(*_*)
Handle a request to advertise services.

connect_or_disconnect_handler(*ryu_event*)
Handle connection or disconnection of a datapath.

Args: ryu_event (ryu.controller.dpset.EventDP): trigger.

desc_stats_reply_handler(*ryu_event*)
Handle OFPDescStatsReply from datapath.

Args: ryu_event (ryu.controller.ofp_event.EventOFPDescStatsReply): trigger.

error_handler(*ryu_event*)
Handle an OFPError from a datapath.

Args: ryu_event (ryu.controller.ofp_event.EventOFPErrorMsg): trigger

exc_logname = 'faucet.exception'

features_handler(*ryu_event*)
Handle receiving a switch features message from a datapath.

Args: ryu_event (ryu.controller.ofp_event.EventOFPStateChange): trigger.

flowremoved_handler(*ryu_event*)
Handle a flow removed event.

Args: ryu_event (ryu.controller.ofp_event.EventOFPFlowRemoved): trigger.

get_config()
FAUCET experimental API: return config for all Valves.

get_tables(*dp_id*)
FAUCET experimental API: return config tables for one Valve.

logname = 'faucet'

metric_update(*_*)
Handle a request to update metrics in the controller.

packet_in_handler(*ryu_event*)
Handle a packet in event from the dataplane.

Args: ryu_event (ryu.controller.event.EventReplyBase): packet in message.

port_status_handler(*ryu_event*)
Handle a port status change event.

Args: ryu_event (ryu.controller.ofp_event.EventOFPPortStatus): trigger.

reconnect_handler(*ryu_event*)
Handle reconnection of a datapath.

Args: ryu_event (ryu.controller.dpset.EventDPReconnected): trigger.

reload_config(*_*)
Handle a request to reload configuration.

resolve_gateways(*_*)
Handle a request to re/resolve gateways.

start()

state_expire(*_*)
Handle a request expire host state in the controller.

faucet.faucet_bgp module

BGP implementation for FAUCET.

class faucet.faucet_bgp.**FaucetBgp**(*logger*, *send_flow_msgs*)
Bases: object

reset(*valves*, *metrics*)
Set up a BGP speaker for every VLAN that requires it.

update_metrics()
Update BGP metrics.

faucet.faucet_experimental_api module

Implement experimental API.

class faucet.faucet_experimental_api.**FaucetExperimentalAPI**(**args*, ***kwargs*)
Bases: object

An experimental API for communicating with Faucet.

Contains methods for interacting with a running Faucet controller from within a RyuApp. This app should be run together with Faucet in the same ryu-manager process.

add_port_acl(*port*, *acl*)
Add an ACL to a port.

add_vlan_acl(*vlan*, *acl*)
Add an ACL to a VLAN.

delete_port_acl(*port*, *acl*)
Delete an ACL from a port.

delete_vlan_acl(*vlan*, *acl*)
Delete an ACL from a VLAN.

get_config()
Get the current running config of Faucet as a python dictionary.

get_tables(*dp_id*)
Get current FAUCET tables as a dict of table name: table no.

is_registered()
Return True if registered and ready to serve API requests.

push_config(*config*)
Push supplied config to FAUCET.

reload_config()
Reload config from config file in FAUCET_CONFIG env variable.

faucet.faucet_metrics module

Implement Prometheus statistics.

class faucet.faucet_metrics.**FaucetMetrics**
Bases: *faucet.prom_client.PromClient*

Container class for objects that can be exported to Prometheus.

reset_dpid(*dp_labels*)
Set all DPID-only counter/gauges to 0.

faucet.fctl module

Report state based on FAUCET/Gauge/Prometheus variables.

faucet.fctl.**main**()

faucet.fctl.**report_label_match_metrics**(*report_metrics*, *metrics*, *nonzero_only=False*, *delim='\t'*, *label_matches=None*)
Text report on a list of Prometheus metrics.

faucet.fctl.**scrape_prometheus**(*endpoints*, *retries=3*)
Scrape a list of Prometheus/FAUCET/Gauge endpoints and aggregate results.

faucet.fctl.**usage**()

faucet.gauge module

RyuApp shim between Ryu and Gauge.

class faucet.gauge.**EventGaugeReconfigure**
Bases: ryu.controller.event.EventBase

Event sent to Gauge to cause config reload.

class faucet.gauge.**Gauge**(**args*, ***kwargs*)
Bases: ryu.base.app_manager.RyuApp

Ryu app for polling Faucet controlled datapaths for stats/state.

It can poll multiple datapaths. The configuration files for each datapath should be listed, one per line, in the file set as the environment variable GAUGE_CONFIG. It logs to the file set as the environment variable GAUGE_LOG,

OFP_VERSIONS = [4]

exc_logname = 'gauge.exception'

flow_stats_reply_handler(*ryu_event*)
Handle flow stats reply event.

Args: ryu_event (ryu.controller.event.EventReplyBase): flow stats event.

handler_connect_or_disconnect(*ryu_event*)
Handle DP dis/connect.

Args: ryu_event (ryu.controller.event.EventReplyBase): DP reconnection.

handler_reconnect(*ryu_event*)
Handle a DP reconnection event.

Args: ryu_event (ryu.controller.event.EventReplyBase): DP reconnection.

logname = 'gauge'

port_stats_reply_handler(*ryu_event*)
Handle port stats reply event.

Args: ryu_event (ryu.controller.event.EventReplyBase): port stats event.

port_status_handler(*ryu_event*)
Handle port status change event.

Args: ryu_event (ryu.controller.event.EventReplyBase): port status change event.

reload_config(*_*)
Handle request for Gauge config reload.

signal_handler(*sigid*, *_*)
Handle signal and cause config reload.

Args: sigid (int): signal received.

start()

faucet.gauge_influx module

Library for interacting with InfluxDB.

class faucet.gauge_influx.**GaugeFlowTableInfluxDBLogger**(*conf*, *logname*, *prom_client*)

Bases: *faucet.gauge_pollers.GaugeFlowTablePoller*, *faucet.gauge_influx.InfluxShipper*

> use faucet Using database faucet > show series where table_id = '0' and in_port = '2' key — flow_byte_count,dp_name=windscale-faucet-1,eth_type=2048,in_port=2,ip_proto=17,priority=9099,table_id=0,udp_dst=53 flow_byte_count,dp_name=windscale-faucet-1,eth_type=2048,in_port=2,ip_proto=6,priority=9098,table_id=0,tcp_dst=53 flow_byte_count,dp_name=windscale-faucet-1,in_port=2,priority=9097,table_id=0 flow_packet_count,dp_name=windscale-faucet-1,eth_type=2048,in_port=2,ip_proto=17,priority=9099,table_id=0,udp_dst=53 flow_packet_count,dp_name=windscale-faucet-1,eth_type=2048,in_port=2,ip_proto=6,priority=9098,table_id=0,tcp_dst=53 flow_packet_count,dp_name=windscale-faucet-1,in_port=2,priority=9097,table_id=0 > select * from flow_byte_count where table_id = '0' and in_port = '2' and ip_proto = '17' and time > now() - 5m name: flow_byte_count time arp_tpa dp_name eth_dst eth_src eth_type icmpv6_type in_port ip_proto ipv4_dst ipv6_dst priority table_id tcp_dst udp_dst value vlan_vid —- ——- ——- ——- ——- ——- ——— ——- ——— ——— ——— ——— ——— ——— ——- ——- —— ——— 1501154797000000000 windscale-faucet-1 2048 2 17 9099 0 53 9414 1501154857000000000 windscale-faucet-1 2048 2 17 9099 0 53 10554 1501154917000000000 windscale-faucet-1 2048 2 17 9099 0 53 10554 1501154977000000000 windscale-faucet-1 2048 2 17 9099 0 53 12164 1501155037000000000 windscale-faucet-1 2048 2 17 9099 0 53 12239

update(*rcv_time*, *dp_id*, *msg*)

class faucet.gauge_influx.**GaugePortStateInfluxDBLogger**(*conf*, *logname*, *prom_client*)

Bases: *faucet.gauge_pollers.GaugePortStateBaseLogger*, *faucet.gauge_influx.InfluxShipper*

> use faucet Using database faucet > precision rfc3339 > select * from port_state_reason where port_name = 'port1.0.1' order by time desc limit 10; name: port_state_reason ——————— time dp_name port_name value 2017-02-21T02:12:29Z windscale-faucet-1 port1.0.1 2 2017-02-21T02:12:25Z windscale-faucet-1 port1.0.1 2 2016-07-27T22:05:08Z windscale-faucet-1 port1.0.1 2 2016-05-25T04:33:00Z windscale-faucet-1 port1.0.1 2 2016-05-25T04:32:57Z windscale-faucet-1 port1.0.1 2 2016-05-25T04:31:21Z windscale-faucet-1 port1.0.1 2 2016-05-25T04:31:18Z windscale-faucet-1 port1.0.1 2 2016-05-25T04:27:07Z windscale-faucet-1 port1.0.1 2 2016-05-25T04:27:04Z windscale-faucet-1 port1.0.1 2 2016-05-25T04:24:53Z windscale-faucet-1 port1.0.1 2

update(*rcv_time*, *dp_id*, *msg*)

class faucet.gauge_influx.**GaugePortStatsInfluxDBLogger**(*conf*, *logname*, *prom_client*)

Bases: *faucet.gauge_pollers.GaugePortStatsPoller*, *faucet.gauge_influx.InfluxShipper*

Periodically sends a port stats request to the datapath and parses and outputs the response.

> use faucet Using database faucet > show measurements name: measurements ——————— bytes_in bytes_out dropped_in dropped_out errors_in packets_in packets_out port_state_reason > precision rfc3339 > select * from packets_out where port_name = 'port1.0.1' order by time desc limit 10; name: packets_out ——————— time dp_name port_name value 2017-03-06T05:21:42Z windscale-faucet-1 port1.0.1 76083431 2017-03-06T05:21:33Z windscale-faucet-1 port1.0.1 76081172 2017-03-06T05:21:22Z windscale-faucet-1 port1.0.1 76078727 2017-03-06T05:21:12Z windscale-faucet-1 port1.0.1 76076612 2017-03-06T05:21:02Z windscale-faucet-1 port1.0.1 76074546 2017-03-06T05:20:52Z windscale-faucet-1 port1.0.1 76072730 2017-03-06T05:20:42Z windscale-faucet-1 port1.0.1 76070528 2017-03-06T05:20:32Z windscale-faucet-1 port1.0.1 76068211 2017-03-06T05:20:22Z windscale-faucet-1 port1.0.1 76065982 2017-03-06T05:20:12Z windscale-faucet-1 port1.0.1 76063941

update(*rcv_time*, *dp_id*, *msg*)

class faucet.gauge_influx.**InfluxShipper**

Bases: object

Convenience class for shipping values to InfluxDB.

Inheritors must have a WatcherConf object as conf.

conf = None

logger = None

static make_point(*tags*, *rcv_time*, *stat_name*, *stat_val*)
Make an InfluxDB point.

make_port_point(*dp_name*, *port_name*, *rcv_time*, *stat_name*, *stat_val*)
Make an InfluxDB point about a port measurement.

ship_error_prefix = 'error shipping points: '

ship_points(*points*)
Make a connection to InfluxDB and ship points.

faucet.gauge_nsodbc module

Library for interacting with ODBC databases.

class faucet.gauge_nsodbc.**GaugeFlowTableDBLogger**(*conf*, *logname*, *prom_client*)
Bases: *faucet.gauge_pollers.GaugeFlowTablePoller*, *faucet.gauge_nsodbc.GaugeNsODBC*

Periodically dumps the current datapath flow table to ODBC DB.

update(*rcv_time*, *dp_id*, *msg*)

class faucet.gauge_nsodbc.**GaugeNsODBC**
Bases: object

Helper class for NSODBC operations

Inheritors must have a WatcherConf object as conf.

conf = None

conn = None

conn_string = None

db_update_counter = None

flow_database = None

refresh_flowdb()

refresh_switchdb()

setup()

switch_database = None

faucet.gauge_pollers module

Library for polling dataplanes for statistics.

class faucet.gauge_pollers.**GaugeFlowTablePoller**(*conf*, *logname*, *prom_client*)
Bases: *faucet.gauge_pollers.GaugeThreadPoller*

Periodically dumps the current datapath flow table as a yaml object.

Includes a timestamp and a reference ($DATAPATHNAME-flowtables). The flow table is dumped as an OFFlowStatsReply message (in yaml format) that matches all flows.

no_response()

send_req()

class faucet.gauge_pollers.**GaugePoller**(*conf*, *logname*, *prom_client*)
Bases: object

Abstraction for a poller for statistics.

static no_response()
Called when a polling cycle passes without receiving a response.

report_dp_status(*dp_status*)
Report DP status.

static running()
Return True if the poller is running.

static send_req()
Send a stats request to a datapath.

static start(*_ryudp*)
Start the poller.

static stop()
Stop the poller.

update(*rcv_time*, *dp_id*, *msg*)
Handle the responses to requests.

Called when a reply to a stats request sent by this object is received by the controller.

It should acknowledge the receipt by setting self.reply_pending to false.

Arguments: rcv_time – the time the response was received dp_id – DP ID msg – the stats reply message

class faucet.gauge_pollers.**GaugePortStateBaseLogger**(*conf*, *logname*, *prom_client*)
Bases: *faucet.gauge_pollers.GaugePoller*

Abstraction for port state poller.

static no_response()
Called when a polling cycle passes without receiving a response.

static send_req()
Send a stats request to a datapath.

class faucet.gauge_pollers.**GaugePortStatsPoller**(*conf*, *logname*, *prom_client*)
Bases: *faucet.gauge_pollers.GaugeThreadPoller*

Periodically sends a port stats request to the datapath and parses and outputs the response.

no_response()

send_req()

class faucet.gauge_pollers.**GaugeThreadPoller**(*conf*, *logname*, *prom_client*)
Bases: *faucet.gauge_pollers.GaugePoller*

A ryu thread object for sending and receiving OpenFlow stats requests.

The thread runs in a loop sending a request, sleeping then checking a response was received before sending another request.

The methods send_req, update and no_response should be implemented by subclasses.

static no_response()
Called when a polling cycle passes without receiving a response.

running()

static send_req()
Send a stats request to a datapath.

start(*ryudp*)

stop()

faucet.gauge_prom module

Prometheus for Gauge.

class faucet.gauge_prom.**GaugePortStatsPrometheusPoller**(*conf*, *logger*, *prom_client*)
Bases: *faucet.gauge_pollers.GaugePortStatsPoller*

Exports port stats to Prometheus.

update(*rcv_time*, *dp_id*, *msg*)

class faucet.gauge_prom.**GaugePrometheusClient**
Bases: *faucet.prom_client.PromClient*

Wrapper for Prometheus client that is shared between all pollers.

metrics = {}

faucet.meter module

Configure meters.

class faucet.meter.**Meter**(*_id*, *dp_id*, *conf*)
Bases: *faucet.conf.Conf*

Implement FAUCET configuration for an OpenFlow meter.

defaults = {'meter_id': None, 'entry': None}

defaults_type = {'meter_id': <class 'int'>, 'entry': <class 'dict'>}

entry = None

entry_msg = None

meter_id = None

name = None

faucet.nsodbc module

This module exposes an api to deal with db operations on no-sql databases. Currently couchdb support is included.

class faucet.nsodbc.**ConnectionCouch**(*conn*, *credentials*)
Bases: object

Connection class.

This class is specific to couchdb operations. For others a new class will be needed (following same standards)

connected_databases()
Return the connected databases of this connection

create(*db_name*)
Create a database. If the database exists, return the same and send a True flag. This way, a connection object will only be created once.

delete(*db_name*)
Delete database specified in the parameter

class faucet.nsodbc.**DatabaseCouch**(*database*)
Bases: object

Database specific class exposing the API.

create_view(*design*, *views*)
This is a couchdb functionality. Helps in creating views needed for querying the database. Input: Design name, view definition

delete_doc(*doc_id*)
Delete document based on the doc id

get_docs(*view_url*, *key*)
Select docs

A view url is used as select query with the key as a where condition

insert_update_doc(*doc*, *update_key=''*)
Insert or update a document For updating, a key has to be provided against which a document will be updated

class faucet.nsodbc.**NsOdbc**
Bases: object

An abstraction layer to make api calls to a non relational database.

Currently the API provided is: connect create get_doc insert_update_doc delete_doc

connect(**conn_string*, ***kwargs*)
Returns a connection object required for further operations Input: connection string or connection parameters Returns: connection object

get_attributes()
Returns API version

faucet.nsodbc.**init_flow_db**(*flow_database*)
Initialize/Refresh flow database Args:

flow_database

faucet.nsodbc.**init_switch_db**(*switch_database*)
Initialize/refresh switch database Args:

switch_database

faucet.nsodbc.**nsodbc_factory**()
factory method to consume the API

faucet.nsodbc.**todict**(*conn_string*, *kwargs*)
Converts the input connection string into a dictionary.

Assumption: Connection string is of the format 'driver=couchdb;server=localhost;uid=database_uid;pwd=database_pwd'

faucet.port module

Port configuration.

class faucet.port.**Port**(*_id*, *dp_id*, *conf=None*)

Bases: *faucet.conf.Conf*

Implement FAUCET configuration for a port.

acl_in = None

check_config()

defaults = {'unicast_flood': True, 'acl_in': None, 'native_vlan': None, 'number':

defaults_types = {'unicast_flood': <class 'bool'>, 'acl_in': (<class 'str'>, <class

dp_id = None

dyn_lacp_up = None

dyn_lacp_updated_time = None

dyn_last_ban_time = None

dyn_last_lacp_pkt = None

dyn_learn_ban_count = 0

dyn_phys_up = False

enabled = None

finalize()

hairpin = None

hosts(*vlans=None*)

Return all hosts this port has learned (on all or specified VLANs).

loop_protect = None

max_hosts = None

mirror = None

mirror_destination = None

name = None

native_vlan = None

number = None

permanent_learn = None

running()

set_defaults()

stack = {}

tagged_vlans = []

to_conf()

unicast_flood = None

vlans()

Return list of all VLANs this port is in.

faucet.prom_client module

Implement Prometheus client.

class faucet.prom_client.**PromClient**
Bases: object

Prometheus client.

REQUIRED_LABELS = ['dp_id', 'dp_name']

running = False

start(*prom_port*, *prom_addr*)
Start webserver if not already running.

faucet.router module

Configure routing between VLANs.

class faucet.router.**Router**(*_id*, *dp_id*, *conf=None*)
Bases: *faucet.conf.Conf*

Implement FAUCET configuration for a router.

check_config()

defaults = {'vlans': None}

defaults_type = {'vlans': <class 'list'>}

vlans = None

faucet.tfm_pipeline module

Parse JSON for TFM based table config.

class faucet.tfm_pipeline.**LoadRyuTables**(*cfgpath*, *pipeline_conf*)
Bases: object

load_tables()

class faucet.tfm_pipeline.**OpenflowToRyuTranslator**(*cfgpath*, *pipeline_conf*)
Bases: object

create_ryu_structure()

faucet.valve module

Implementation of Valve learning layer 2/3 switch.

class faucet.valve.**ArubaValve**(*dp*, *logname*)
Bases: *faucet.valve.TfmValve*

Valve implementation that uses OpenFlow send table features messages.

DEC_TTL = False

PIPELINE_CONF = 'aruba_pipeline.json'

class faucet.valve.**TfmValve**(*dp*, *logname*)
Bases: *faucet.valve.Valve*

Valve implementation that uses OpenFlow send table features messages.

PIPELINE_CONF = 'tfm_pipeline.json'

SKIP_VALIDATION_TABLES = ()

switch_features(*msg*)

class faucet.valve.**Valve**(*dp*, *logname*)
Bases: object

Generates the messages to configure a datapath as a l2 learning switch.

Vendor specific implementations may require sending configuration flows. This can be achieved by inheriting from this class and overwriting the function switch_features.

DEC_TTL = True

L3 = False

add_route(*vlan*, *ip_gw*, *ip_dst*)
Add route to VLAN routing table.

advertise()
Called periodically to advertise services (eg. IPv6 RAs).

base_prom_labels = None

control_plane_handler(*pkt_meta*)
Handle a packet probably destined to FAUCET's route managers.

For example, next hop resolution or ICMP echo requests.

Args: pkt_meta (PacketMeta): packet for control plane.

Returns: list: OpenFlow messages, if any.

datapath_connect(*discovered_up_port_nums*)
Handle Ryu datapath connection event and provision pipeline.

Args: discovered_up_port_nums (list): datapath ports that are up as ints.

Returns: list: OpenFlow messages to send to datapath.

datapath_disconnect()
Handle Ryu datapath disconnection event.

del_route(*vlan*, *ip_dst*)
Delete route from VLAN routing table.

flow_timeout(*table_id*, *match*)

get_config_dict()

lacp_down(*port*)

lacp_handler(*pkt_meta*)
Handle a LACP packet.

We are a currently a passive, non-aggregateable LACP partner.

Args: pkt_meta (PacketMeta): packet for control plane.

Returns: list: OpenFlow messages, if any.

lacp_up(*port*)

ofchannel_log(*ofmsgs*)
Log OpenFlow messages in text format to debugging log.

parse_rcv_packet(*in_port*, *vlan_vid*, *eth_type*, *data*, *orig_len*, *pkt*, *eth_pkt*)
Parse a received packet into a PacketMeta instance.

Args: in_port (int): port packet was received on. vlan_vid (int): VLAN VID of port packet was received on. eth_type (int): Ethernet type of packet. data (bytes): Raw packet data. orig_len (int): Original length of packet. pkt (ryu.lib.packet.packet): parsed packet received. ekt_pkt (ryu.lib.packet.ethernet): parsed Ethernet header.

Returns: PacketMeta instance.

port_add(*port_num*)
Handle addition of a single port.

Args: port_num (list): list of port numbers.

Returns: list: OpenFlow messages, if any.

port_delete(*port_num*)

port_status_handler(*port_no*, *reason*, *port_status*)

ports_add(*port_nums*, *cold_start=False*)
Handle the addition of ports.

Args: port_num (list): list of port numbers. cold_start (bool): True if configuring datapath from scratch.

Returns: list: OpenFlow messages, if any.

ports_delete(*port_nums*)
Handle the deletion of ports.

Args: port_nums (list): list of port numbers.

Returns: list: OpenFlow messages, if any.

rcv_packet(*other_valves*, *pkt_meta*)
Handle a packet from the dataplane (eg to re/learn a host).

The packet may be sent to us also in response to FAUCET initiating IPv6 neighbor discovery, or ARP, to resolve a nexthop.

Args: other_valves (list): all Valves other than this one. pkt_meta (PacketMeta): packet for control plane.

Return: list: OpenFlow messages, if any.

recent_ofmsgs = <queue.Queue object>

reload_config(*new_dp*)
Reload configuration new_dp.

Following config changes are currently supported:

- **Port config: support all available configs (e.g. native_vlan, acl_in)** & change operations (add, delete, modify) a port
- **ACL config:support any modification, currently reload all rules** belonging to an ACL
- VLAN config: enable, disable routing, etc...

Args: new_dp (DP): new dataplane configuration.

Returns:

tuple of: cold_start (bool): whether cold starting. ofmsgs (list): OpenFlow messages.

resolve_gateways()
Call route managers to re/resolve gateways.

Returns: list: OpenFlow messages, if any.

state_expire()
Expire controller caches/state (e.g. hosts learned).

Expire state from the host manager only; the switch does its own flow expiry.

Return: list: OpenFlow messages, if any.

switch_features(*_msg*)
Send configuration flows necessary for the switch implementation.

Arguments: msg – OFPSwitchFeatures msg sent from switch.

Vendor specific configuration should be implemented here.

update_config_metrics(*metrics*)
Update gauge/metrics for configuration.

metrics (FaucetMetrics): container of Prometheus metrics.

update_metrics(*metrics*)
Update Gauge/metrics.

metrics (FaucetMetrics or None): container of Prometheus metrics.

class faucet.valve.**ValveLogger**(*logger*, *dp_id*)
Bases: object

debug(*log_msg*)

error(*log_msg*)

info(*log_msg*)

warning(*log_msg*)

faucet.valve.**valve_factory**(*dp*)
Return a Valve object based dp's hardware configuration field.

Args: dp (DP): DP instance with the configuration for this Valve.

faucet.valve_acl module

Compose ACLs on ports.

faucet.valve_acl.**build_acl_entry**(*rule_conf*, *acl_allow_inst*, *meters*, *port_num=None*, *vlan_vid=None*)

faucet.valve_acl.**build_acl_ofmsgs**(*acls*, *acl_table*, *acl_allow_inst*, *highest_priority*, *meters*, *exact_match*, *port_num=None*, *vlan_vid=None*)

faucet.valve_acl.**build_output_actions**(*output_dict*)
Implement actions to alter packet/output.

faucet.valve_acl.**push_vlan**(*vlan_vid*)
Push a VLAN tag with optional selection of eth type.

faucet.valve_acl.**rewrite_vlan**(*output_dict*)
Implement actions to rewrite VLAN headers.

faucet.valve_flood module

Manage flooding to ports on VLANs.

class faucet.valve_flood.**ValveFloodManager**(*flood_table*, *flood_priority*, *use_group_table*, *groups*)

Bases: object

Implement dataplane based flooding for standalone dataplanes.

FLOOD_DSTS = ((True, None, None), (False, '01:80:c2:00:00:00', 'ff:ff:ff:00:00:00'), (

build_flood_rules(*vlan*, *modify=False*)
Add flows to flood packets to unknown destinations on a VLAN.

static edge_learn_port(*_other_valves*, *pkt_meta*)
Possibly learn a host on a port.

Args: other_valves (list): All Valves other than this one. pkt_meta (PacketMeta): PacketMeta instance for packet received.

Returns: port to learn host on.

class faucet.valve_flood.**ValveFloodStackManager**(*flood_table*, *flood_priority*, *use_group_table*, *groups*, *stack*, *stack_ports*, *dp_shortest_path_to_root*, *shortest_path_port*)

Bases: *faucet.valve_flood.ValveFloodManager*

Implement dataplane based flooding for stacked dataplanes.

build_flood_rules(*vlan*, *modify=False*)
Add flows to flood packets to unknown destinations on a VLAN.

edge_learn_port(*other_valves*, *pkt_meta*)
Possibly learn a host on a port.

Args: other_valves (list): All Valves other than this one. pkt_meta (PacketMeta): PacketMeta instance for packet received.

Returns: port to learn host on, or None.

faucet.valve_host module

Manage host learning on VLANs.

class faucet.valve_host.**ValveHostFlowRemovedManager**(*logger*, *ports*, *vlans*, *eth_src_table*, *eth_dst_table*, *learn_timeout*, *learn_jitter*, *learn_ban_timeout*, *low_priority*, *host_priority*)

Bases: *faucet.valve_host.ValveHostManager*

Trigger relearning on flow removed notifications.

NOTE: not currently reliable.

expire_hosts_from_vlan(*_vlan*, *_now*)

flow_timeout(*table_id*, *match*)

learn_host_timeouts(*port*)
Calculate flow timeouts for learning on a port.

class faucet.valve_host.**ValveHostManager**(*logger*, *ports*, *vlans*, *eth_src_table*, *eth_dst_table*, *learn_timeout*, *learn_jitter*, *learn_ban_timeout*, *low_priority*, *host_priority*)

Bases: object

ban_rules(*pkt_meta*)
Limit learning to a maximum configured on this port/VLAN.

Args: pkt_meta: PacketMeta instance.

Returns: list: OpenFlow messages, if any.

build_port_out_inst(*vlan*, *port*, *port_number=None*)
Return instructions to output a packet on a given port.

delete_host_from_vlan(*eth_src*, *vlan*)
Delete a host from a VLAN.

expire_hosts_from_vlan(*vlan*, *now*)
Expire hosts from VLAN cache.

flow_timeout(*_table_id*, *_match*)

learn_host_on_vlan_port_flows(*port*, *vlan*, *eth_src*, *delete_existing*, *src_rule_idle_timeout*, *src_rule_hard_timeout*, *dst_rule_idle_timeout*)
Return flows that implement learning a host on a port.

learn_host_on_vlan_ports(*port*, *vlan*, *eth_src*, *delete_existing=True*)
Learn a host on a port.

learn_host_timeouts(*port*)
Calculate flow timeouts for learning on a port.

faucet.valve_of module

Utility functions to parse/create OpenFlow messages.

faucet.valve_of.**apply_actions**(*actions*)
Return instruction that applies action list.

Args: actions (list): list of OpenFlow actions.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPInstruction: instruction of actions.

faucet.valve_of.**apply_meter**(*meter_id*)
Return instruction to apply a meter.

faucet.valve_of.**barrier**()
Return OpenFlow barrier request.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPBarrierRequest: barrier request.

faucet.valve_of.**bucket**(*weight=0*, *watch_port=4294967295*, *watch_group=4294967295*, *actions=None*)
Return a group action bucket with provided actions.

faucet.valve_of.**build_match_dict**(*in_port=None*, *vlan=None*, *eth_type=None*, *eth_src=None*, *eth_dst=None*, *eth_dst_mask=None*, *ipv6_nd_target=None*, *icmpv6_type=None*, *nw_proto=None*, *nw_src=None*, *nw_dst=None*)

faucet.valve_of.**controller_pps_meteradd**(*datapath=None*, *pps=0*)
Add a PPS meter towards controller.

faucet.valve_of.**controller_pps_meterdel**(*datapath=None*)
Delete a PPS meter towards controller.

faucet.valve_of.**dec_ip_ttl**()
Return OpenFlow action to decrement IP TTL.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPActionDecNwTtl: decrement IP TTL.

faucet.valve_of.**desc_stats_request**(*datapath=None*)
Query switch description.

faucet.valve_of.**devid_present**(*vid*)
Return VLAN VID without VID_PRESENT flag set.

Args: vid (int): VLAN VID with VID_PRESENT.

Returns: int: VLAN VID.

faucet.valve_of.**faucet_async**(*datapath=None*)
Return async message config for FAUCET.

faucet.valve_of.**faucet_config**(*datapath=None*)
Return switch config for FAUCET.

faucet.valve_of.**flood_tagged_port_outputs**(*ports*, *in_port*, *exclude_ports=None*)
Return list of actions necessary to flood to list of tagged ports.

faucet.valve_of.**flood_untagged_port_outputs**(*ports*, *in_port*, *exclude_ports=None*)
Return list of actions necessary to flood to list of untagged ports.

faucet.valve_of.**flowmod**(*cookie*, *command*, *table_id*, *priority*, *out_port*, *out_group*, *match_fields*, *inst*, *hard_timeout*, *idle_timeout*, *flags=0*)

faucet.valve_of.**gauge_async**(*datapath=None*)
Return async message config for Gauge.

faucet.valve_of.**goto_table**(*table*)
Return instruction to goto table.

Args: table (ValveTable): table to goto.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPInstruction: goto instruction.

faucet.valve_of.**group_act**(*group_id*)
Return an action to run a group.

faucet.valve_of.**group_flood_buckets**(*ports*, *untagged*)

faucet.valve_of.**groupadd**(*datapath=None*, *type_=0*, *group_id=0*, *buckets=None*)
Add a group.

faucet.valve_of.**groupadd_ff**(*datapath=None*, *group_id=0*, *buckets=None*)
Add a fast failover group.

faucet.valve_of.**groupdel**(*datapath=None*, *group_id=4294967292*)
Delete a group (default all groups).

faucet.valve_of.**groupmod**(*datapath=None*, *type_=0*, *group_id=0*, *buckets=None*)
Modify a group.

faucet.valve_of.**groupmod_ff**(*datapath=None*, *group_id=0*, *buckets=None*)
Modify a fast failover group.

faucet.valve_of.**ignore_port**(*port_num*)
Return True if FAUCET should ignore this port.

Args: port_num (int): switch port.

Returns: bool: True if FAUCET should ignore this port.

faucet.valve_of.**is_flowdel**(*ofmsg*)
Return True if flow message is a FlowMod and a delete.

Args: ofmsg: ryu.ofproto.ofproto_v1_3_parser message.

Returns: bool: True if is a FlowMod delete/strict.

faucet.valve_of.**is_flowmod**(*ofmsg*)
Return True if flow message is a FlowMod.

Args: ofmsg: ryu.ofproto.ofproto_v1_3_parser message.

Returns: bool: True if is a FlowMod

faucet.valve_of.**is_groupadd**(*ofmsg*)
Return True if OF message is a GroupMod and command is add.

Args: ofmsg: ryu.ofproto.ofproto_v1_3_parser message.

Returns: bool: True if is a GroupMod add

faucet.valve_of.**is_groupdel**(*ofmsg*)
Return True if OF message is a GroupMod and command is delete.

Args: ofmsg: ryu.ofproto.ofproto_v1_3_parser message.

Returns: bool: True if is a GroupMod delete

faucet.valve_of.**is_groupmod**(*ofmsg*)
Return True if OF message is a GroupMod.

Args: ofmsg: ryu.ofproto.ofproto_v1_3_parser message.

Returns: bool: True if is a GroupMod

faucet.valve_of.**match**(*match_fields*)
Return OpenFlow matches from dict.

Args: match_fields (dict): match fields and values.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPMatch: matches.

faucet.valve_of.**match_from_dict**(*match_dict*)

faucet.valve_of.**meteradd**(*meter_conf*)
Add a meter based on YAML configuration.

faucet.valve_of.**meterdel**(*datapath=None*, *meter_id=4294967295*)
Delete a meter (default all meters).

faucet.valve_of.**output_controller**(*max_len=128*)
Return OpenFlow action to packet in to the controller.

Args: max_len (int): max number of bytes from packet to output.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPActionOutput: packet in action.

faucet.valve_of.**output_in_port**()
Return OpenFlow action to output out input port.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPActionOutput.

faucet.valve_of.**output_port**(*port_num*, *max_len=0*)
Return OpenFlow action to output to a port.

Args: port_num (int): port to output to. max_len (int): maximum length of packet to output (default no maximum).

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPActionOutput: output to port action.

faucet.valve_of.**packetout**(*port_num*, *data*)
Return OpenFlow action to packet out to dataplane from controller.

Args: port_num (int): port to output to. data (str): raw packet to output.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPActionOutput: packet out action.

faucet.valve_of.**pop_vlan**()
Return OpenFlow action to pop outermost Ethernet 802.1Q VLAN header.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPActionPopVlan: Pop VLAN.

faucet.valve_of.**push_vlan_act**(*vlan_vid*, *eth_type=33024*)
Return OpenFlow action list to push Ethernet 802.1Q header with VLAN VID.

Args: vid (int): VLAN VID

Returns: list: actions to push 802.1Q header with VLAN VID set.

faucet.valve_of.**set_eth_dst**(*eth_dst*)
Return action to set destination Ethernet MAC address.

Args: eth_src (str): destination Ethernet MAC address.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPActionSetField: set field action.

faucet.valve_of.**set_eth_src**(*eth_src*)
Return action to set source Ethernet MAC address.

Args: eth_src (str): source Ethernet MAC address.

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPActionSetField: set field action.

faucet.valve_of.**set_vlan_vid**(*vlan_vid*)
Set VLAN VID with VID_PRESENT flag set.

Args: vid (int): VLAN VID

Returns: ryu.ofproto.ofproto_v1_3_parser.OFPActionSetField: set VID with VID_PRESENT.

faucet.valve_of.**table_features**(*body*)

faucet.valve_of.**valve_flowreorder**(*input_ofmsgs*)
Reorder flows for better OFA performance.

faucet.valve_of.**valve_match_vid**(*value*)

faucet.valve_of.**vid_present**(*vid*)
Return VLAN VID with VID_PRESENT flag set.

Args: vid (int): VLAN VID

Returns: int: VLAN VID with VID_PRESENT.

faucet.valve_packet module

Utility functions for parsing and building Ethernet packet/contents.

class faucet.valve_packet.**PacketMeta**(*data*, *orig_len*, *pkt*, *eth_pkt*, *port*, *valve_vlan*, *eth_src*, *eth_dst*, *eth_type*)

Bases: object

Original, and parsed Ethernet packet metadata.

ETH_TYPES_PARSERS = {2048: (4, <function ipv4_parseable>, <class 'ryu.lib.packet.ipv4

ip_ver()
Return IP version number.

packet_complete()
True if we have the complete packet.

reparse(*max_len*)
Reparse packet using data up to the specified maximum length.

reparse_all()
Reparse packet with all available data.

reparse_ip(*eth_type*, *payload=0*)
Reparse packet with specified IP header type and optionally payload.

faucet.valve_packet.**arp_reply**(*vid*, *eth_src*, *eth_dst*, *src_ip*, *dst_ip*)
Return an ARP reply packet.

Args: vid (int or None): VLAN VID to use (or None). eth_src (str): Ethernet source address. eth_dst (str): destination Ethernet MAC address. src_ip (ipaddress.IPv4Address): source IPv4 address. dst_ip (ipaddress.IPv4Address): destination IPv4 address.

Returns: ryu.lib.packet.arp: serialized ARP reply packet.

faucet.valve_packet.**arp_request**(*vid*, *eth_src*, *src_ip*, *dst_ip*)
Return an ARP request packet.

Args: vid (int or None): VLAN VID to use (or None). eth_src (str): Ethernet source address. src_ip (ipaddress.IPv4Address): source IPv4 address. dst_ip (ipaddress.IPv4Address): requested IPv4 address.

Returns: ryu.lib.packet.arp: serialized ARP request packet.

faucet.valve_packet.**build_pkt_header**(*vid*, *eth_src*, *eth_dst*, *dl_type*)
Return an Ethernet packet header.

Args: vid (int or None): VLAN VID to use (or None). eth_src (str): source Ethernet MAC address. eth_dst (str): destination Ethernet MAC address. dl_type (int): EtherType.

Returns: ryu.lib.packet.ethernet: Ethernet packet with header.

faucet.valve_packet.**echo_reply**(*vid*, *eth_src*, *eth_dst*, *src_ip*, *dst_ip*, *data*)
Return an ICMP echo reply packet.

Args: vid (int or None): VLAN VID to use (or None). eth_src (str): Ethernet source address. eth_dst (str): destination Ethernet MAC address. src_ip (ipaddress.IPv4Address): source IPv4 address. dst_ip (ipaddress.IPv4Address): destination IPv4 address.

Returns: ryu.lib.packet.icmp: serialized ICMP echo reply packet.

faucet.valve_packet.**icmpv6_echo_reply**(*vid*, *eth_src*, *eth_dst*, *src_ip*, *dst_ip*, *hop_limit*, *id_*, *seq*, *data*)
Return IPv6 ICMP echo reply packet.

Args: vid (int or None): VLAN VID to use (or None). eth_src (str): source Ethernet MAC address. eth_dst (str): destination Ethernet MAC address. src_ip (ipaddress.IPv6Address): source IPv6 address. dst_ip (ipaddress.IPv6Address): destination IPv6 address. hop_limit (int): IPv6 hop limit. **id_** (int): identifier for echo reply. seq (int): sequence number for echo reply. data (str): payload for echo reply.

Returns: ryu.lib.packet.ethernet: Serialized IPv6 ICMP echo reply packet.

faucet.valve_packet.**ip_header_size**(*eth_type*)
Return size of a packet header with specified ether type.

faucet.valve_packet.**ipv4_parseable**(*ip_header_data*)
Return True if an IPv4 packet we could parse.

faucet.valve_packet.**ipv6_link_eth_mcast**(*dst_ip*)
Return an Ethernet multicast address from an IPv6 address.

See RFC 2464 section 7.

Args: dst_ip (ipaddress.IPv6Address): IPv6 address.

Returns: str: Ethernet multicast address.

faucet.valve_packet.**ipv6_solicited_node_from_ucast**(*ucast*)
Return IPv6 solicited node multicast address from IPv6 unicast address.

See RFC 3513 section 2.7.1.

Args: ucast (ipaddress.IPv6Address): IPv6 unicast address.

Returns: ipaddress.IPv6Address: IPv6 solicited node multicast address.

faucet.valve_packet.**lacp_reqreply**(*eth_src*, *actor_system*, *actor_key*, *actor_port*, *partner_system*, *partner_key*, *partner_port*, *partner_system_priority*, *partner_port_priority*, *partner_state_defaulted*, *partner_state_expired*, *partner_state_timeout*, *partner_state_collecting*, *partner_state_distributing*, *partner_state_aggregation*, *partner_state_synchronization*, *partner_state_activity*)
Return a LACP frame.

Args: eth_src (str): source Ethernet MAC address. actor_system (str): actor system ID (MAC address) actor_key (int): actor's LACP key assigned to this port. actor_port (int): actor port number. partner_system (str): partner system ID (MAC address) partner_key (int): partner's LACP key assigned to this port. partner_port (int): partner port number. partner_system_priority (int): partner's system priority. partner_port_priority (int): partner's port priority. partner_state_defaulted (int): 1 if partner reverted to defaults. partner_state_expired (int): 1 if partner thinks LACP expired. partner_state_timeout (int): 1 if partner has short timeout. partner_state_collecting (int): 1 if partner receiving on this link. partner_state_distributing (int): 1 if partner transmitting on this link. partner_state_aggregation (int): 1 if partner can aggregate this link. partner_state_synchronization (int): 1 if partner will use this link. partner_state_activity (int): 1 if partner actively sends LACP.

Returns: ryu.lib.packet.ethernet: Ethernet packet with header.

faucet.valve_packet.**mac_addr_is_unicast**(*mac_addr*)
Returns True if mac_addr is a unicast Ethernet address.

Args: mac_addr (str): MAC address.

Returns: bool: True if a unicast Ethernet address.

faucet.valve_packet.**mac_byte_mask**(*mask_bytes=0*)
Return a MAC address mask with n bytes masked out.

faucet.valve_packet.**nd_advert**(*vid*, *eth_src*, *eth_dst*, *src_ip*, *dst_ip*)
Return IPv6 neighbor avertisement packet.

Args: vid (int or None): VLAN VID to use (or None). eth_src (str): source Ethernet MAC address. eth_dst (str): destination Ethernet MAC address. src_ip (ipaddress.IPv6Address): source IPv6 address. dst_ip (ipaddress.IPv6Address): destination IPv6 address.

Returns: ryu.lib.packet.ethernet: Serialized IPv6 neighbor discovery packet.

faucet.valve_packet.**nd_request**(*vid*, *eth_src*, *src_ip*, *dst_ip*)
Return IPv6 neighbor discovery request packet.

Args: vid (int or None): VLAN VID to use (or None). eth_src (str): source Ethernet MAC address. src_ip (ipaddress.IPv6Address): source IPv6 address. dst_ip (ipaddress.IPv6Address): requested IPv6 address.

Returns: ryu.lib.packet.ethernet: Serialized IPv6 neighbor discovery packet.

faucet.valve_packet.**parse_eth_pkt**(*pkt*)
Return parsed Ethernet packet.

Args: pkt (ryu.lib.packet.packet): packet received from dataplane.

Returns: ryu.lib.packet.ethernet: Ethernet packet.

faucet.valve_packet.**parse_lacp_pkt**(*pkt*)
Return parsed LACP packet.

Args: pkt (ryu.lib.packet.packet): packet received from dataplane.

Returns: ryu.lib.packet.lacp: LACP packet.

faucet.valve_packet.**parse_packet_in_pkt**(*data*, *max_len*)
Parse a packet received via packet in from the dataplane.

Args: data (bytearray): packet data from dataplane. max_len (int): max number of packet data bytes to parse.

Returns: ryu.lib.packet.ethernet: Ethernet packet. int: VLAN VID. int: Ethernet type of packet (inside VLAN)

faucet.valve_packet.**parse_vlan_pkt**(*pkt*)
Return parsed VLAN header.

Args: pkt (ryu.lib.packet.packet): packet received from dataplane.

Returns: ryu.lib.packet.vlan: VLAN header.

faucet.valve_packet.**router_advert**(*_vlan*, *vid*, *eth_src*, *eth_dst*, *src_ip*, *dst_ip*, *vips*, *pi_flags=6*)
Return IPv6 ICMP echo reply packet.

Args: _vlan (VLAN): VLAN instance. vid (int or None): VLAN VID to use (or None). eth_src (str): source Ethernet MAC address. eth_dst (str): dest Ethernet MAC address. src_ip (ipaddress.IPv6Address): source IPv6 address. vips (list): prefixes (ipaddress.IPv6Address) to advertise. pi_flags (int): flags to set in prefix information field (default set A and L)

Returns: ryu.lib.packet.ethernet: Serialized IPv6 ICMP RA packet.

faucet.valve_route module

Valve IPv4/IPv6 routing implementation.

class faucet.valve_route.**NextHop**(*eth_src*, *now*)
Bases: object

Describes a directly connected (at layer 2) nexthop.

class faucet.valve_route.**ValveIPv4RouteManager**(*logger*, *arp_neighbor_timeout*, *max_hosts_per_resolve_cycle*, *max_host_fib_retry_count*, *max_resolve_backoff_time*, *proactive_learn*, *dec_ttl*, *fib_table*, *vip_table*, *eth_src_table*, *eth_dst_table*, *flood_table*, *route_priority*, *routers*, *use_group_table*, *groups*)

Bases: *faucet.valve_route.ValveRouteManager*

Implement IPv4 RIB/FIB.

CONTROL_ETH_TYPES = (2048, 2054)

ETH_TYPE = 2048

ICMP_TYPE = 1

IPV = 4

control_plane_handler(*pkt_meta*)

class faucet.valve_route.**ValveIPv6RouteManager**(*logger*, *arp_neighbor_timeout*, *max_hosts_per_resolve_cycle*, *max_host_fib_retry_count*, *max_resolve_backoff_time*, *proactive_learn*, *dec_ttl*, *fib_table*, *vip_table*, *eth_src_table*, *eth_dst_table*, *flood_table*, *route_priority*, *routers*, *use_group_table*, *groups*)

Bases: *faucet.valve_route.ValveRouteManager*

Implement IPv6 FIB.

CONTROL_ETH_TYPES = (34525,)

ETH_TYPE = 34525

ICMP_TYPE = 58

IPV = 6

advertise(*vlan*)

control_plane_handler(*pkt_meta*)

class faucet.valve_route.**ValveRouteManager**(*logger*, *arp_neighbor_timeout*, *max_hosts_per_resolve_cycle*, *max_host_fib_retry_count*, *max_resolve_backoff_time*, *proactive_learn*, *dec_ttl*, *fib_table*, *vip_table*, *eth_src_table*, *eth_dst_table*, *flood_table*, *route_priority*, *routers*, *use_group_table*, *groups*)

Bases: object

Base class to implement RIB/FIB.

CONTROL_ETH_TYPES = None

ETH_TYPE = None

ICMP_TYPE = None

IPV = None

MAX_LEN = 128

add_faucet_vip(*vlan*, *faucet_vip*)

add_host_fib_route_from_pkt(*pkt_meta*)
Add a host FIB route given packet from host.

Args: pkt_meta (PacketMeta): received packet.

Returns: list: OpenFlow messages.

add_route(*vlan*, *ip_gw*, *ip_dst*)
Add a route to the RIB.

Args: vlan (vlan): VLAN containing this RIB. ip_gw (ipaddress.ip_address): IP address of nexthop. ip_dst (ipaddress.ip_network): destination IP network.

Returns: list: OpenFlow messages.

advertise(*vlan*)

control_plane_handler(*pkt_meta*)

del_route(*vlan*, *ip_dst*)
Delete a route from the RIB.

Only one route with this exact destination is supported.

Args: vlan (vlan): VLAN containing this RIB. ip_dst (ipaddress.ip_network): destination IP network.

Returns: list: OpenFlow messages.

resolve_gateways(*vlan*, *now*)
Re/resolve all gateways.

Args: vlan (vlan): VLAN containing this RIB/FIB. now (float): seconds since epoch.

Returns: list: OpenFlow messages.

resolve_gw_on_vlan(*vlan*, *faucet_vip*, *ip_gw*)

faucet.valve_table module

Abstraction of an OF table.

class faucet.valve_table.**ValveGroupEntry**(*table*, *group_id*, *buckets*)
Bases: object

Abstraction for a single OpenFlow group entry.

add()
Return flows to add this entry to the group table.

delete()
Return flow to delete an existing group entry.

modify()
Return flow to modify an existing group entry.

update_buckets(*buckets*)

class faucet.valve_table.**ValveGroupTable**
Bases: object

Wrap access to group table.

delete_all()
Delete all groups.

entries = {}

get_entry(*group_id*, *buckets*)

static group_id_from_str(*key_str*)
Return a group ID based on a string key.

class faucet.valve_table.**ValveTable**(*table_id*, *name*, *restricted_match_types*, *flow_cookie*, *notify_flow_removed=False*)
Bases: object

Wrapper for an OpenFlow table.

flowcontroller(*match=None*, *priority=None*, *inst=None*, *max_len=96*)
Add flow outputting to controller.

flowdel(*match=None*, *priority=None*, *out_port=4294967295*, *strict=False*)
Delete matching flows from a table.

flowdrop(*match=None*, *priority=None*, *hard_timeout=0*)
Add drop matching flow to a table.

flowmod(*match=None*, *priority=None*, *inst=None*, *command=0*, *out_port=0*, *out_group=0*, *hard_timeout=0*, *idle_timeout=0*)
Helper function to construct a flow mod message with cookie.

match(*in_port=None*, *vlan=None*, *eth_type=None*, *eth_src=None*, *eth_dst=None*, *eth_dst_mask=None*, *ipv6_nd_target=None*, *icmpv6_type=None*, *nw_proto=None*, *nw_src=None*, *nw_dst=None*)
Compose an OpenFlow match rule.

faucet.valve_util module

Utility functions for FAUCET.

faucet.valve_util.**btos**(*b_str*)
Return byte array/string as string.

faucet.valve_util.**dpid_log**(*dpid*)
Log a DP ID as hex/decimal.

faucet.valve_util.**get_bool_setting**(*name*)
Return True if setting is a non-zero int.

faucet.valve_util.**get_logger**(*logname*, *logfile*, *loglevel*, *propagate*)
Create and return a logger object.

faucet.valve_util.**get_setting**(*name*)
Returns value of specified configuration setting.

faucet.valve_util.**get_sys_prefix**()
Returns an additional prefix for log and configuration files when used in a virtual environment

faucet.valve_util.**kill_on_exception**(*logname*)
decorator to ensure functions will kill ryu when an unhandled exception occurs

faucet.valve_util.**stat_config_files**(*config_hashes*)
Return dict of a subset of stat attributes on config files.

faucet.vlan module

VLAN configuration.

class faucet.vlan.**HostCacheEntry**(*eth_src*, *port*, *cache_time*)
Bases: object

class faucet.vlan.**VLAN**(*_id*, *dp_id*, *conf=None*)
Bases: *faucet.conf.Conf*

Implement FAUCET configuration for a VLAN.

acl_in = None

add_cache_host(*eth_src*, *port*, *cache_time*)

add_tagged(*port*)

add_untagged(*port*)

bgp_as = None

bgp_local_address = None

bgp_neighbor_addresses = []

bgp_neighbor_as = None

bgp_neighbour_addresses = []

bgp_neighbour_as = None

bgp_port = None

bgp_routerid = None

bgp_server_addresses = []

cached_host(*eth_src*)

cached_host_on_port(*eth_src*, *port*)
Return host cache entry if host in cache and on specified port.

cached_hosts_on_port(*port*)
Return all hosts learned on a port.

check_config()

clear_cache_hosts_on_port(*port*)
Clear all hosts learned on a port.

defaults = {'unicast_flood': True, 'bgp_server_addresses': ['0.0.0.0', '::'], 'route

defaults_types = {'unicast_flood': <class 'bool'>, 'bgp_server_addresses': <class 'l

dp_id = None

dyn_faucet_vips_by_ipv = None

dyn_host_cache = None

dyn_learn_ban_count = 0

dyn_neigh_cache_by_ipv = None

dyn_routes_by_ipv = None

expire_cache_hosts(*now*, *learn_timeout*)
Expire stale host entries.

faucet_mac = None

faucet_vips = None

faucet_vips_by_ipv(*ipv*)
Return list of VIPs with specified IP version on this VLAN.

flood_pkt(*packet_builder*, **args*)

flood_ports(*configured_ports*, *exclude_unicast*)

from_connected_to_vip(*src_ip*, *dst_ip*)
Return True if src_ip in connected network and dst_ip is a VIP.

Args: src_ip (ipaddress.ip_address): source IP. dst_ip (ipaddress.ip_address): destination IP

Returns: True if local traffic for a VIP.

get_ports()
Return list of all ports on this VLAN.

hairpin_ports()
Return all ports with hairpin enabled.

host_cache
Return host (L2) cache for this VLAN.

hosts_count()
Return number of hosts learned on this VLAN.

ip_in_vip_subnet(*ipa*)
Return faucet_vip if IP in same IP network as a VIP on this VLAN.

ips_in_vip_subnet(*ips*)
Return True if all IPs are on same subnet as VIP on this VLAN.

ipvs()
Return list of IP versions configured on this VLAN.

is_faucet_vip(*ipa*)
Return True if IP is a VIP on this VLAN.

lags()
Return dict of LAGs mapped to member ports.

max_hosts = None

mirror_destination_ports()
Return list of ports that are mirrored to, on this VLAN.

mirrored_ports()
Return list of ports that are mirrored on this VLAN.

name = None

neigh_cache_by_ipv(*ipv*)
Return neighbor cache for specified IP version on this VLAN.

port_is_tagged(*port*)
Return True if port number is an tagged port on this VLAN.

port_is_untagged(*port*)
Return True if port number is an untagged port on this VLAN.

proactive_arp_limit = None

proactive_nd_limit = None

reset_host_cache()

routes = None

routes_by_ipv(*ipv*)
Return route table for specified IP version on this VLAN.

set_defaults()

tagged = None

tagged_flood_ports(*exclude_unicast*)

unicast_flood = None

untagged = None

untagged_flood_ports(*exclude_unicast*)

vid = None

static vid_valid(*vid*)
Return True if VID valid.

faucet.watcher module

Gauge watcher implementations.

class faucet.watcher.**GaugeFlowTableLogger**(*conf*, *logname*, *prom_client*)
Bases: *faucet.gauge_pollers.GaugeFlowTablePoller*

Periodically dumps the current datapath flow table as a yaml object.

Includes a timestamp and a reference ($DATAPATHNAME-flowtables). The flow table is dumped as an OFFlowStatsReply message (in yaml format) that matches all flows.

optionally the output can be compressed by setting compressed: true in the config for this watcher

update(*rcv_time*, *dp_id*, *msg*)

class faucet.watcher.**GaugePortStateLogger**(*conf*, *logname*, *prom_client*)
Bases: *faucet.gauge_pollers.GaugePortStateBaseLogger*

Abstraction for port state logger.

static no_response()
Called when a polling cycle passes without receiving a response.

static send_req()
Send a stats request to a datapath.

update(*rcv_time*, *dp_id*, *msg*)

class faucet.watcher.**GaugePortStatsLogger**(*conf*, *logname*, *prom_client*)
Bases: *faucet.gauge_pollers.GaugePortStatsPoller*

Abstraction for port statistics logger.

update(*rcv_time*, *dp_id*, *msg*)

faucet.watcher.**watcher_factory**(*conf*)
Return a Gauge object based on type.

Arguments: gauge_conf – a GaugeConf object with the configuration for this valve.

faucet.watcher_conf module

Gauge watcher configuration.

class faucet.watcher_conf.**WatcherConf**(*_id*, *dp_id*, *conf*, *prom_client*)
Bases: *faucet.conf.Conf*

Gauge watcher configuration.

add_db(*db_conf*)
Add database config to this watcher.

add_dp(*dp*)
Add a datapath to this watcher.

all_dps = None

db = None

defaults = {'influx_db': 'faucet', 'influx_user': '', 'file': None, 'driver': '',

dp = None

prom_client = None

Module contents

CHAPTER 3

Quick References

3.1 Frequently Asked Questions

3.1.1 How are packet-ins handled when a message is generated through table-miss flow entry?

Faucet adds explicit rules for unmatched packets.

3.1.2 Are group actions supported in Faucet?

Yes, just not by default currently. Set the `group_table` option to `True` on a datapath to enable group output actions.

3.1.3 Does Faucet send any multi-part requests? If so, please provide sample use cases

Gauge uses multi-part messages for the stats collection (flow table stats and port stats).

3.1.4 Does Faucet install table-miss entry?

Yes.

3.1.5 Does Faucet clear all all switch table entries on connection?

Faucet gives all entries a specific cookie, and it clears all entries with that cookie. I.e., it clears entries added by itself but not anyone else.

3.1.6 Does Faucet install fresh set of table entries on connection and re-connection?

Yes.

3.1.7 Does Faucet installed flows support priority? How is this defined - who get higher priority than the other and why?

Yes, priority is necessary for a number of things. Example: there are higher priority rules for packets with a known source address, and lower ones to send those packets to the controller.

3.1.8 Is there a gui for generating a YAML file?

No.

3.1.9 Should Faucet detect Management, OF controller ports and gateway ports on the switch or pure OF only ports where hosts are connected?

Out of scope for Faucet as it is currently.

3.1.10 If another controller is connected to the switch in addition to Faucet, what happens to Faucet?

Faucet identifies its own flows using a cookie value, if the other controller doesn't use the same cookie value there shouldn't be a problem (provided the rules don't conflict in a problematic way)

3.1.11 If another controller connected to switch changes role (master, slave, equal) on the switch, what happens to Faucet?

Shouldn't be an issue, if another controller is the master then my understanding is Faucet wouldnt be able to install any flows however?

3.1.12 Does Faucet send LLDP packets?

No.

3.1.13 Some switches always send VLAN info in packet_in messages and some don't. How does Faucet handle this?

Packets should have VLANs pushed before being sent to the controller.

3.1.14 Is there a event handler registered to detect if flows on the switch change?

No.

3.1.15 Does Faucet use auxiliary connections?

No.

3.1.16 Does Faucet support L2.5 (MPLS, etc.)?

No.

3.1.17 Stats - what does Faucet collect (flow count, etc)?

Gauge collects port stats and takes a full flow-table dump periodically.

3.1.18 How do I use Gauge?

Give Gauge a list of Faucet yaml config files and it will poll them for stats (as specified in the config file).

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W